hitex DEVELOPMENT TOOLS

SafeTI™-Hitex Safety Kit(HSK)

User Manual of the Hitex Safety Kit

User Manual

40100, V1.2, 2013-06-17

User Manual

Edition 2013-06-17

Published by:
Hitex Development Tools GmbH
Greschbachstr. 12, 76229 Karlsruhe, Germany
© 2013 Hitex Development Tools GmbH
All Rights Reserved.

Legal Disclaimer

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the product, Hitex Development Tools GmbH hereby disclaims any and all warranties and liabilities of any kind, including without limitation, warranties of non-infringement of intellectual property rights of any third party.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Hitex Office (www.hitex.com).

Document history			
Date	**Version**	**Modified by**	**Modification description**
2013-05-22	1.0	Wenz	Initial Draft
2013-06-10	1.1	Wenz	Reworked
2013-06-17	1.2	Wenz, Arnaout, Sander	Review points considered Ready to release

We ask for your comments

Is there any information in this document that you feel is wrong, unclear or missing?
Your feedback will help to continuously improve the quality of this document.
Please send your comments (including a reference to this document) to:
application@hitex.de

Table of Contents

1 Abbreviations, definitions and scope of document 10
1.1 Abbreviations 10
1.2 Definitions 11
1.3 Scope of document 11
1.4 Related documents 11

2 Introduction 12
2.1 Purpose of the document 13
2.2 Outline of the document 13

3 Installation 14
3.1 System requirements 14
3.2 Quick start 14
3.2.1 How to update your kit 15
3.2.1.1 Updating your kit with UniFlash 15
3.3 Tools installation 19

4 System description 21
4.1 System components 21
4.2 System behavior 22
4.2.1 Startup, power on 22
4.2.1.1 Safety application firmware 23
4.2.1.2 Control monitor device firmware 23
4.2.2 External watchdog TPS 24
4.2.3 Normal operation 24
4.2.3.1 Safety application firmware 24

5 Hardware Description 25
5.1 Hitex Safety Kit Features 25
5.2 Physical Description 26
5.2.1 External Interfaces 26
5.2.1.1 JTAG Connector 26
5.2.1.2 TI DRV8312 controlCARD Interface 26
5.2.1.3 CAN Connector 28
5.2.1.4 DIAGOUT Jumper 28
5.2.1.5 USB Connector 28
5.2.1.6 Power Supply Connector 29
5.2.2 Display 29
5.2.3 LEDs 29
5.2.4 Push Buttons 30
5.2.5 Sensors 30
5.2.5.1 Digital Accelerometer 30
5.2.5.2 Temperature Sensor 31
5.2.5.3 Potentiometer 31
5.2.6 Test Points 31

6 HSK Monitor Graphical User Interface (GUI) 35
6.1 Main Window 35
6.1.1 Main Viewing Area (1) 35
6.1.2 Navigation Bar (2) 36
6.1.3 Hyperlinks (3) 37
6.1.4 Button “Stop recording” (4) 37
6.1.5 Status Bar (5) 37
6.2 GUI Pages 38
6.2.1 Overview Page 38
6.2.2 Validation & Profiling Page 39
6.2.3 Global Settings Page 42
6.2.3.1 Safety loop 43
6.2.3.2 System load 43

6.2.3.3 ESM Settings 44
6.2.3.4 TPS6538x Settings 45
6.2.4 Diagnostic Settings Page 46
6.2.4.1 Group - Self Test Controller (STC) 47
6.2.4.2 Group - Power state controller (PSCON) 47
6.2.4.3 Group - Programmable Memory BIST (pBIST) 47
6.2.4.4 Group - Tests on SRAM 47
6.2.4.5 Group - Tests on Flash 48
6.2.4.6 Group – CRC calculation 48
6.2.4.7 Group – Efuse Static Configuration 48
6.2.4.8 Group – CPU compare module (CCM-R4F) 48
6.2.4.9 Group – ADC 48
6.2.5 Monitoring Page 49
6.2.6 Application Page 49
6.2.7 User Commands Page 50

7 Fault injection 52
7.1 Data flow 52
7.2 System behavior, categorization 52
7.2.1 Signals and their meaning 52
7.2.2 Times measured 54
7.2.3 Error servicing mechanism 54
7.2.4 Faults affecting the ESM (group 1 and group 2) 54
7.2.5 Faults which lead to an abort 55
7.2.6 Faults injected on Power rails to TPS 56
7.2.7 Faults injected on Power rails and reset lines to SDUT 57
7.2.8 Faults affecting the TPS Q&A protocol 57
7.2.9 Faults detected by software application 58
7.3 Faults 59

8 Profiling 69
8.1 Data flow 69
8.1.1 Understanding what is measured 69
8.1.2 Special Considerations 70
8.2 Profiling Tests 70
8.2.1 Specific details for the tests 70
8.2.1.1 Dedicated tests calling the self-test in the safety library directly 70
8.2.1.2 Run time parameterized tests 70
8.2.2 Profiling Tests list 72
8.2.3 Profiling full safety task 75

9 Application example demonstration 76
9.1 Temperature sensor 76
9.2 Accelerometer 77
9.3 Onboard display 78
9.4 Push buttons 78
9.5 LEDs 78
9.6 User Commands (Template task) 79
9.7 Task monitoring 79

10 The safety application 80
10.1 Considerations before you start 80
10.2 Tooling 80
10.2.1 Import projects into CCS 82
10.3 Safety application firmware 83
10.3.1 Directory structure 83
10.4 Architecture 84
10.5 Kit application 84
10.5.1 SafeRTOS 85
10.5.2 Source files 86
10.5.3 Task overview 86

10.5.4 HSA_CMD_Handler Task 87
10.5.5 HSA_FI_Handler 88
10.5.6 HSA_SafetyLibServer 89
10.5.6.1 Run time self-test execution 90
10.5.6.2 Profiling measurement 90
10.5.7 HSA_HMI_SensorAndHMI 91
10.5.8 HSA_WDService_Task 92
10.5.9 HSA_DisplayTask 93
10.5.10 HSA_Template_Task 94
10.5.11 Parameters 94
10.5.11.1 Global settings handling 95
10.5.12 Typical data flow example for a fault injection 96
10.5.13 Typical data flow example for a profiling measurement 97

11 Troubleshooting 98

12 Appendix A: References 99

List of figures

Figure 2-1: SafeTI™-HSK....12
Figure 3-1 Directory structure....14
Figure 3-2 Setup Type installation screen....15
Figure 3-3 Processor selection installation screen....16
Figure 3-4 Emulator selection screen....16
Figure 3-5 Target Configuration selection screen....17
Figure 3-6 Target configurations....17
Figure 3-7 Program process selection....18
Figure 3-8 Download applications....18
Figure 3-9 Tools DVD setup screen....19
Figure 3-10 Tools DVD software installation....19
Figure 4-1: Main components of the kit....21
Figure 4-2: Startup Flow Diagram....23
Figure 5-1 Block diagram of the kit....25
Figure 5-2 Input Power Supply Polarization....29
Figure 5-3 Test points available on HSK....33
Figure 6-1 GUI Main Window after start-up (edited)....35
Figure 6-2 GUI Main Window after connection....37
Figure 6-3 Overview Page....38
Figure 6-4 Validation & Profiling Page....40
Figure 6-5 Fault selection, configuration and injection....41
Figure 6-6 Test profiling in the Validation & Profiling Page....42
Figure 6-7 Global Settings Page....43
Figure 6-8 Diagnostic Settings Page....46
Figure 6-9 Monitoring Page....49
Figure 6-10 Application Page....50
Figure 6-11 User Commands Page....51
Figure 7-1 Fault injection signals....52
Figure 7-2 Data flow of the SDUT no recovery from fault....54
Figure 7-3 Data flow of the SDUT recovery from fault....54
Figure 7-4 Category 1 fault; Lockstep PSCON....55
Figure 7-5 Data flow of the SDUT....55
Figure 7-6 Category 2 fault; PMA on PSCON....56
Figure 7-7 Category 3 fault; Under voltage on VBAT....56
Figure 7-8 Category 4 fault; Disturb CoreVCC 1.2V....57
Figure 7-9 Category 5 fault; MCU sends data outside allowed window....58
Figure 7-10 category 6 fault; CRC check....59
Figure 7-11 category 7 fault; CRC check at boot time....59
Figure 7-12 Faults....68
Figure 8-1: Data flow of C&M device....69
Figure 8-2: Data flow of safety device....69
Figure 8-3: Profile Timing invalid....70
Figure 8-4: Data flow of safety device....71
Figure 8-5: Parameter set for the pBIST test....71
Figure 9-1: March of temperature....76
Figure 9-2: Acceleration move....77
Figure 9-3: Task monitor....79
Figure 10-1: Project properties – Linked resources....81
Figure 10-2: Project properties – CCS Build....81
Figure 10-3: Import dialogue....82
Figure 10-4: Import projects....82
Figure 10-5: Directories....83
Figure 10-6: SAFW Architecture top level....84
Figure 10-7: kit application source files....86
Figure 10-8: Architecture Task Level....86
Figure 10-9: Generic structure of tasks in the Safety Kit Application....87
Figure 10-10: Command handler task data flow....88

Figure 10-11: Fault Injection handler task data flow 88
Figure 10-12: Safety library server task data flow 89
Figure 10-13: Profiling measurement data flow 90
Figure 10-14: Profiling full safety task data flow 91
Figure 10-15: Sensor task data flow 92
Figure 10-16: Watchdog service task data flow 93
Figure 10-17: Source code extract configuration settings data storage 95
Figure 10-18: data flow of the handling for the Parameter settings 96
Figure 10-19: data flow of fault injection 96
Figure 10-20: data flow profiling 97

List of tables

Table 1-1: Abbreviations 11
Table 1-2: Definitions 11
Table 3-1 Provided Tools 20
Table 5-1 External Interfaces 26
Table 5-2 JTAG connector pin assignment 26
Table 5-3 Motor control interface pin assignment 28
Table 5-4 CAN connector pin assignment 28
Table 5-5 DIAGOUT jumper pin assignment 28
Table 5-6 Display interface 29
Table 5-7 LED indicators 30
Table 5-8 User-programmable LEDs 30
Table 5-9 User-programmable push buttons 30
Table 5-10 Digital accelerometer interface 31
Table 5-11 Temperature sensor interface 31
Table 5-12 Potentiometer interface 31
Table 5-13 Test points between Safety MCU and TPS65381 32
Table 5-14 Test points between Safety MCU and C&M MCU 33
Table 5-15 Mapping between test points and GUI signals 34
Table 6-1 Buttons in the Navigation Bar 36
Table 6-2 Information within the System flow part of the Overview Page 39
Table 6-3 Signals within the System flow window part during fault injection 41
Table 6-4: ESM error configuration 45
Table 6-5 TPS Settings 46
Table 8-1 Profiling tests 75
Table 9-1 User-programmable LEDs 78
Table 10-1 Command handler task properties 87
Table 10-2 Command handler message queues 87
Table 10-3 Fault handler task properties 89
Table 10-4 Fault handler message queues 89
Table 10-5 Safety library server task properties 90
Table 10-6 Sensor handler task properties 91
Table 10-7 Sensor handler message queues 91
Table 10-8 Watchdog server task properties 92
Table 10-9 Display task properties 93
Table 10-10 Display message queues 93
Table 10-11 used files for onboard display api 94
Table 10-12 Template task properties 94
Table 10-13 Template task message queues 94
Table 11-1: Troubleshooting 98

1 Abbreviations, definitions and scope of document

1.1 Abbreviations

Abbreviation	Comment
ADC	Analog to Digital Converter
API	Application Programming Interface
C&M device	Control And Monitoring Device
CCM-R4	CPU Compare Module for Cortex™ R4F
CCS	Code Composer Studio
ENDRV	Enable Driver (Output pin from TPS)
ECC	Error Correction Code
ESM	Error Signaling Module
FEE	Flash Emulated EEPROM
FMC	Flash Memory Controller
HALCoGen	Hardware Abstraction Layer Code Generator
HCLK	Primary CPU and memory subsystem Clock
LCD	Liquid Crystal Display
LED	Light Emitting Diode
MCU	Microcontroller Unit
MIBSPI	Multi-Buffered Serial Peripheral Interface
NTC	Negative Temperature Coefficient Thermistor
OSAL	Operating System Abstraction Layer
PBIST	Programmable Built In Self Test
PMIC	Power management integrated circuit
PMM	Power Management Module
PSCON	Power State Controller
Q&A	Question And Answer
RGB LED	Red, Green, Blue LED
RTI	Real Time Interrupt (Module)
SAFW	Safety Application Firmware
SDUT	Safety Device Under Test
SDL	SafeTI™ Diagnostics Library
SIMO	SPI Connection, Slave In Master Out
SOMI	SPI Connection, Slave Out Master In
SPI	Serial Protocol Interface
STC	Self Test Controller
TCRAM	Tightly Coupled Random Access Memory

Abbreviation	Comment
UM	User Manual
VCLK	Primary peripheral Clock
VIM	Vectored Interrupt Manager (Module)

Table 1-1: Abbreviations

1.2 Definitions

Definition	Comment
Error Detection Time	This is the time duration from when a fault is injected (fault injection signal) until the error is detected (ERR detection signal).
Monitor App	Monitor application. This is the firmware running on the C&M device.
Safety App	Safety application. This is the firmware running on the SDUT
TPS	A separate controller which provides the power supply to the SDUT. It also incorporates a watchdog.

Table 1-2: Definitions

1.3 Scope of document

This document contains the user documentation for the SafeTI™-HSK kit.

1.4 Related documents

Modifications to any of the following documents can have an impact on this document:

- Quick Start Guide.
- Safety application source code documentation.
- Control/Monitor application source code documentation
- HSK-Monitor GUI source code documentation

Referenced documents (for information only, no requirements):

- TMS570LS31x/21x 16/32-Bit RISC Flash Microcontroller Technical Reference Manual; Literature Number: SPNU499A; November 2012
- RM48 16/32-Bit RISC Flash Microcontroller Technical Reference Manual; Literature Number: SPNU503A; November 2012
- TMS570LS3137 16/32-Bit RISC Flash Microcontroller Datasheet; Literature Number: SPNS162A; November 2012
- RM48 16/32-Bit RISC Flash Microcontroller Datasheet; Literature Number: SPNS174; September 2011
- TPS65381-Q1 Datasheet; Literature Number: SLVSBC4; May 2012
- Safety Manual for TMS570LS31x and TMS570LS21x Hercules™ ARM® Safety Critical Microcontrollers User's Guide; Literature Number: SPNU511B; April 2013
- TMS470/570 Platform F035 Flash API Reference Guide Version 1.06; Literature Number: SPNU493C; April 2012
- nowECC Generation Tool Version 2.17 User's Guide; Literature Number; SPNU491B; August 2011
- SAFE**RTOS** Datasheet
- SAFE**RTOS** User's Manual for the Code Composer Studio TMS570 MPU Product Variant; Report Number: 34-172-MAN-1-005-006; Issue Number: 1.0; 12 May 2011

2 Introduction

The SafeTI™-HSK is a valuable evaluation tool to explore how to achieve functional safety with SafeTI™ Hercules microcontrollers and TPS6538x PMIC devices. The kit provides a hardware reference design and a software application framework to enable an application developer to build a safety application using TI's Hercules™ ARM® safety microcontrollers.

The hardware for the kit consists of two Hercules™ ARM® Safety microcontroller devices, one which acts as a Safety Device Under Test (SDUT), and the second as a Control and Monitoring Device (C&M device). The SDUT MCU is available in two variants: one with the TMS570LS3137 and one with the RM48L952. The C&M device is always a RM48L952. The TI TPS65381 multi rail power supply chip, which is a companion PMIC for Hercules™ safety MCUs, provides the power supply for the SDUT. This device also has an integrated watchdog to be able to supervise the SDUT.

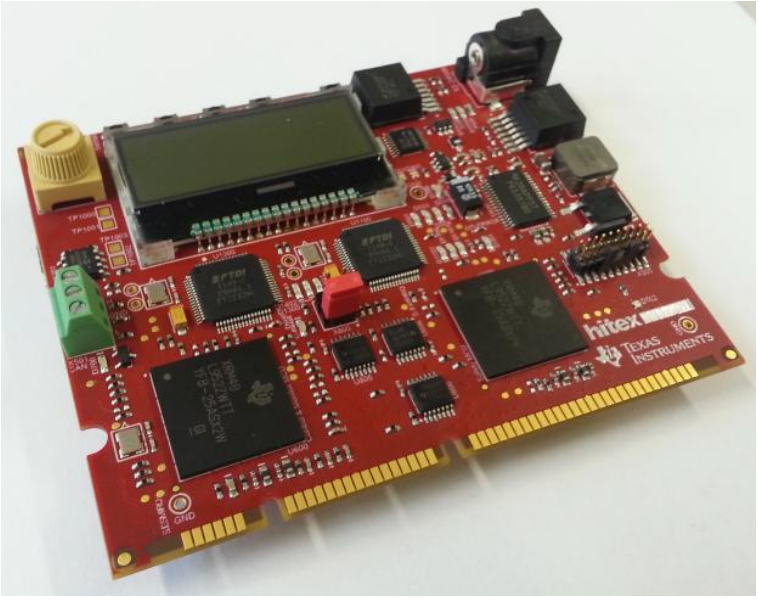

Figure 2-1: SafeTI™-HSK

There are two firmware projects available with the Code Composer Studio IDE (Eclipse based), which can be evaluated: A safety application runs on the SDUT, while a monitor application runs on the C&M Device. Both firmware projects are based on safeRTOS, which is a real-time operating system for use in safety critical designs, and are delivered with this kit (full source code except the code of safeRTOS).

The safety application firmware additionally includes two libraries: TI's SafeTI™ Diagnostic Library, which provides interfaces to run the self-tests/safety diagnostics, and another library for serving the external watchdog (TPS6538X).

The software package is delivered on two DVDs:

- SafeTI™-HSK DVD is for users who want to explore the example application
- The Tool DVD is needed when the example safety application needs to be modified / extended and debugged.

A graphical user interface called HSK-Monitor is included in the kit, which is a Windows application that communicates with the board via USB. Among others, it provides the capability to inject faults to the SDUT and observe its behavior. It also allows to profile runtime self-tests.

With these features, an application developer can design different runtime self-test configurations to design their safety system.

The safety application firmware is explained in chapter 10 of this manual in further detail.

2.1 Purpose of the document

The purpose of this document is to explain how to use the SafeTI™-HSK kit for evaluating the safety features of the Hercules controllers and the TPS6538X PMIC in conjunction with the example application software.
For users who want to investigate the safety application software further, the document explains how the application can be extended, modified and rebuilt with Code Composer Studio.

2.2 Outline of the document

Chapter 3 Installation
Explains the handling and installation instructions of the two DVD packages delivered with this kit.
Chapter 4 System description
Contains the system description explaining the major system components.
Chapter 5 Hardware Description
Contains a description of the hardware.
Chapter 6 HSK Monitor Graphical User Interface (GUI)
Contains the HSK-Monitor (GUI) description. All the GUI window pages are explained.
Chapter 7 Fault injection
Explains the fault injection functionality in detail.
Chapter 8 Profiling
Explains the profiling functionality in detail.
Chapter 9 Application example demonstration
Contains a description of the application examples.
Chapter 10 The safety application
Explains the safety application firmware framework, the included libraries together with major instances.
Chapter 11 Troubleshooting
Explains how to troubleshoot problems with the kit.

3 Installation

The SafeTI™-HSK includes two DVDs: the SafeTI™-HSK DVD for the kit installation, and a Tool DVD for tools installation.

- The SafeTI™-HSK DVD is sufficient if you only intend to evaluate the kit.
- The Tools DVD is required only if the safety application will be investigated/modified/extended.

3.1 System requirements

Software
Windows 7, Windows Vista or Windows XP
Windows .Net framework 4.0 or later
Code Composer 5.3
FTDI drivers (included in the quick start installation and will be installed automatically)

Licenses
For evaluation purpose of the kit no product licenses are necessary.

- SafeRTOS runs with an evaluation license. It is a runtime license, which limits the use of the kit to eight hours continuously. After a power-on cycle, evaluation can be restarted again.
- Code Composer Studio needs no license, since it is connected to a TI controller.

3.2 Quick start

For a quick start with the kit, the SafeTI™-HSK DVD must be installed. Insert the DVD and start with setup.exe. During installation, the HSK-Monitor GUI and associated drivers, an update tool for the firmware, documentation and the firmware projects for the SDUT and C&M devices are installed. After a successful installation, you will find the following directories on your PC.

- Documentation: Quick Start Guide (QSG), HSK - User Manual, PCB-schematics
- Drivers: contains the FTDI drivers for USB communication between SafeTI™-HSK and the PC
- Firmware application:
  - One directory for the control-and-monitor application
  - One directory for the safety application
  - One directory containing the UniFlash utility for updating the kit.
- GUI: contains all data required for the graphical command user interface “HSK-Monitor.exe”.
- Release notes
-

The following figure shows this structure:

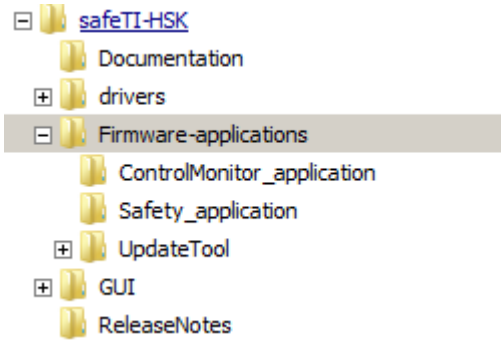

Figure 3-1 Directory structure

3.2.1 *How to update your kit*

The GUI checks at startup for any updates on the Hitex SafeTI™-HSK update website. It also checks whether the firmware in the kit is up-to-date with the newest one found in the installation directory (..\safeTI-HSK\Firmware-applications\...). Downloading and installing the updates is completely optional, nevertheless highly recommended.

To update your kit with a newer firmware, it is required that you have either Code Composer Studio or the UniFlash utility (The CCS UniFlash is provided with the SafeTI™-HSK installation) installed on your PC. The Code Composer Studio method is beyond the scope of this document, and hence the recommended method is using CCS UniFlash.

3.2.1.1 Updating your kit with UniFlash

The first step is to have CCS UniFlash installed with the correct options on your PC. The following describes the steps needed to correctly install CCS UniFlash, which can be found under "..\safeTI-HSK\Firmware-applications\UpdateTool\.

- Start **uniflash_setup_2.0.0.00013.exe** installation and follow the instructions.
- Accept the license agreement and select an installation directory.
- Select the setup type "Custom".

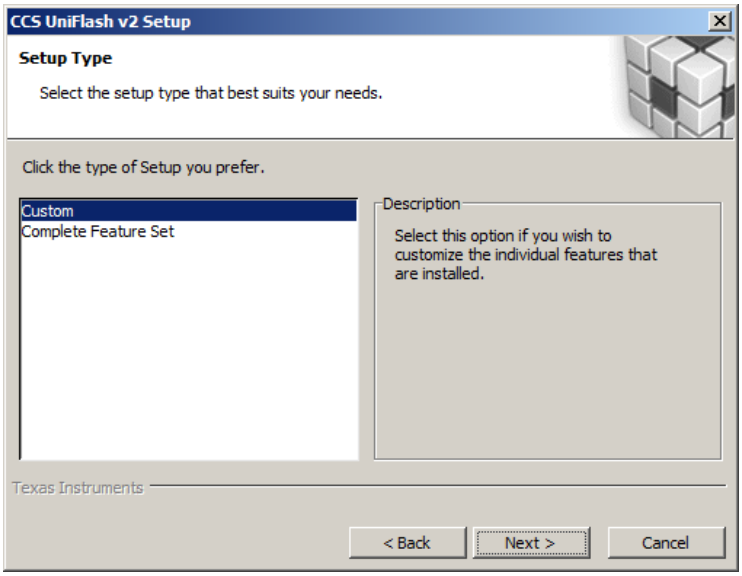
CCS UniFlash v2 Setup
Setup Type
Select the setup type that best suits your needs.
Click the type of Setup you prefer.
Custom
Complete Feature Set
Description
Select this option if you wish to customize the individual features that are installed.
Texas Instruments
< Back
Next >
Cancel

Figure 3-2 Setup Type installation screen

- Select the processor architecture Cortex-R4F MCUs

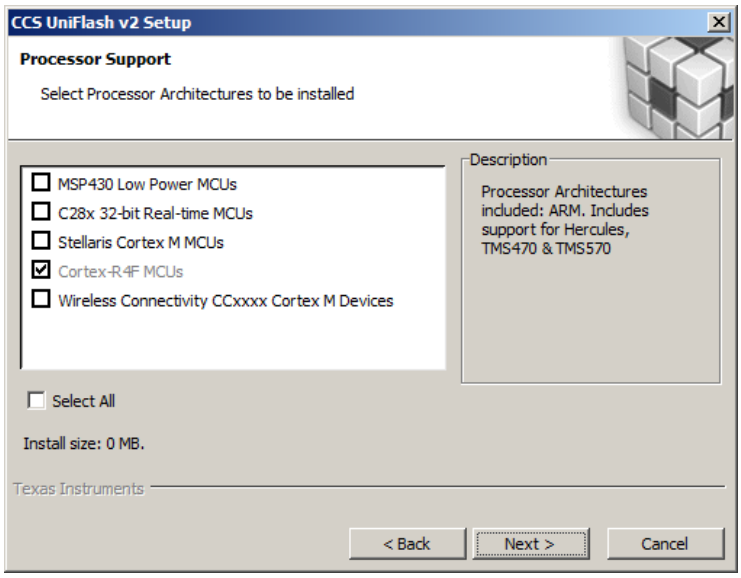

Figure 3-3 Processor selection installation screen

- Select the emulators you are using. Select at least XDS100 Class Emulator support.

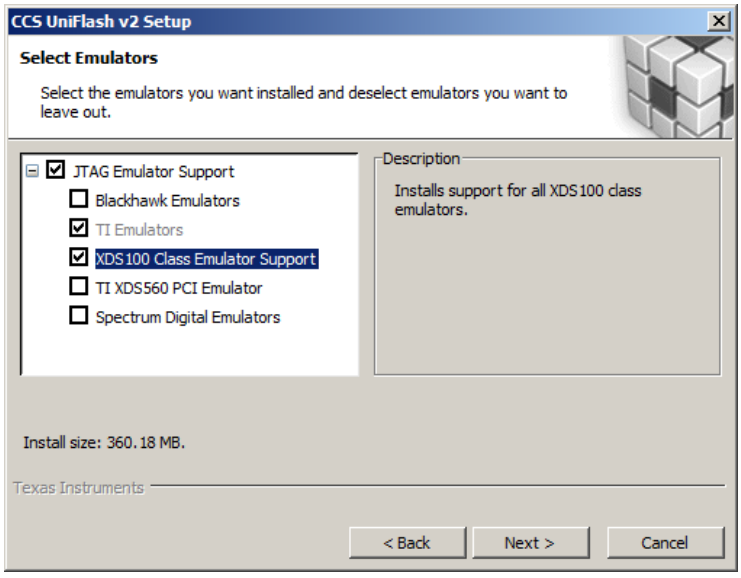

Figure 3-4 Emulator selection screen

- Wait until the installation has finished and start the tool.

The next step is to download the new firmware into the kit. Since the kit features two controllers onboard and two controller variants for the SDUT, selecting the correct controller to update is done by loading a configuration file specific to that controller. The following steps describe the process of updating the kit's firmware

- Load a configuration by selecting File > Open Target Configuration (*.ccxml) File

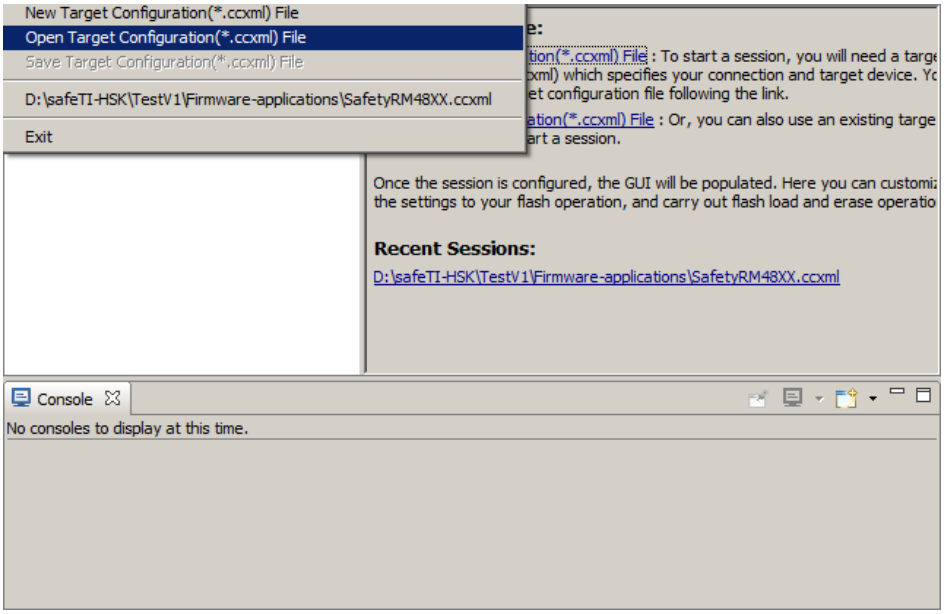
New Target Configuration(*.ccxml) File
Open Target Configuration(*.ccxml) File
Save Target Configuration(*.ccxml) File
D:\safeTI-HSK\TestV1\Firmware-applications\SafetyRM48XX.ccxml
Exit
Once the session is configured, the GUI will be populated. Here you can customi
the settings to your flash operation, and carry out flash load and erase operatio
Recent Sessions:
D:\safeTI-HSK\TestV1\Firmware-applications\SafetyRM48XX.ccxml
Console
No consoles to display at this time.

Figure 3-5 Target Configuration selection screen

- Navigate to the directory "..\safeTI-HSK\Firmware-applications\" and select the required configuration file
  - To update the firmware of the C&M device, select MonitorRM48L950.ccxml.
  - To update the firmware of an RM48L952 SDUT, select SafetyRM48XX.ccxml.
  - To update the firmware a TMS570LS3137 SDUT, select SafetyTMS570LS3137.ccxml

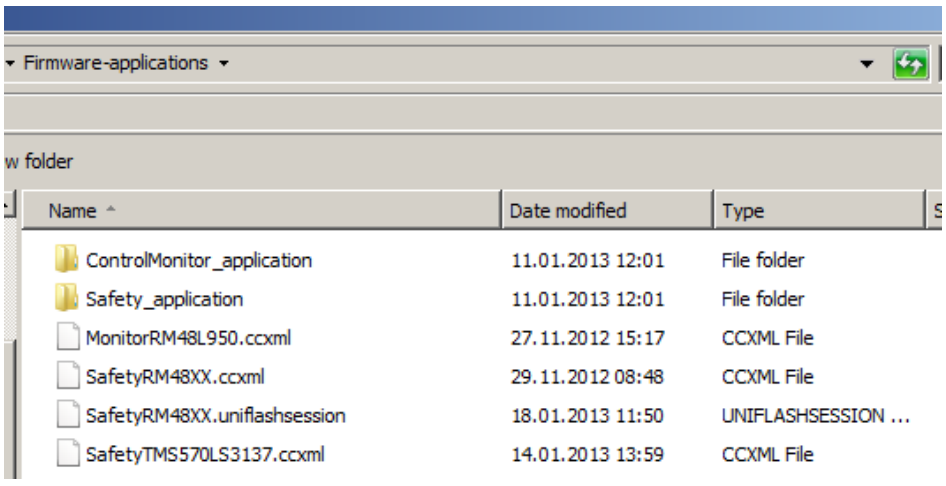

Figure 3-6 Target configurations

- Select “Programs” from the settings tree on the left pane and then click on the “Add” button to select the firmware to be downloaded.
    - The image file for the C&M device can be found under: ..\safeTI-HSK\Firmware-applications\ControlMonitor_application\HSK_Monitor_application.out
    - The image file for an RM48L952 SDUT can be found under: ..\safeTI-HSK\Firmware-applications\Safety_application\HSK_Safety_Application_LE.out
    - The image file for a TMS570LS3137 SDUT can be found under: ..\safeTI-HSK\Firmware-applications\Safety_application\HSK_Safety_Application_BE.out

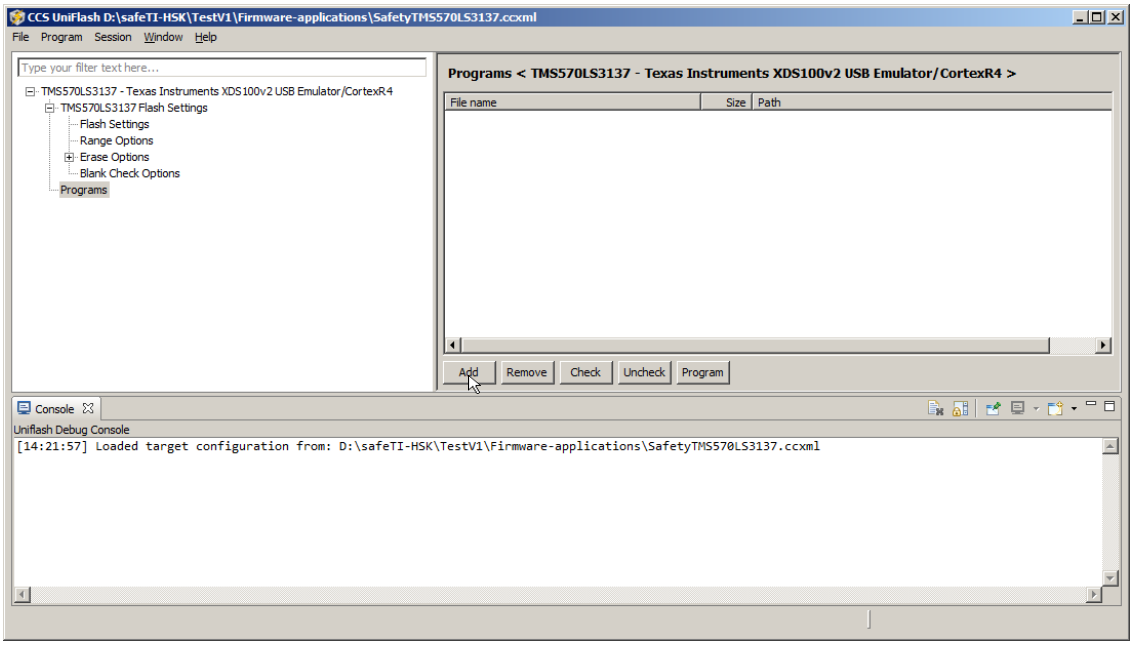

Figure 3-7 Program process selection

- Select the image and click the “Program” button to start the download.

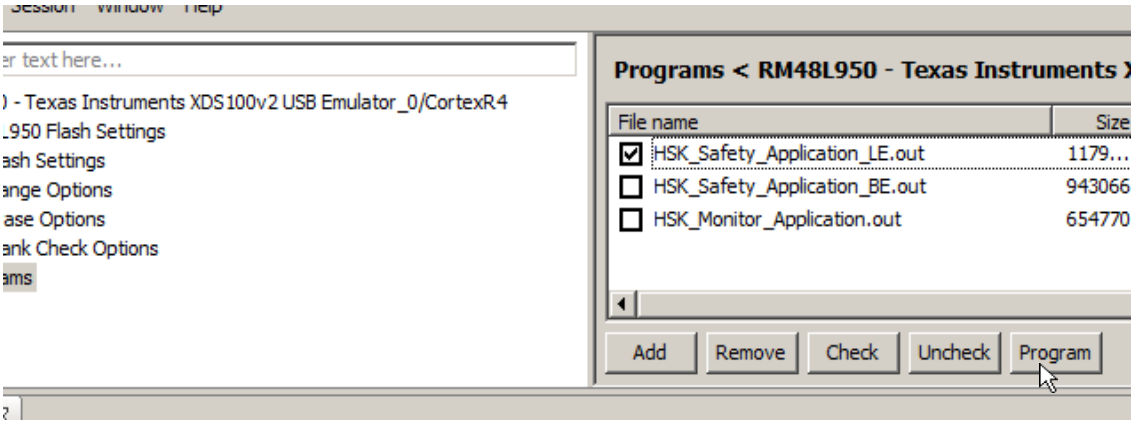

Figure 3-8 Download applications

- After a successful update, power-cycle the board to restart the application

3.3 Tools installation

For users who want to explore or even modify/extend the source code of the safety application some tools are required and others are helpful.

Note: Code Composer Studio (CCS) is essential to debug/develop the safety application further.

After starting with setup.exe on the DVD, the following window is shown. Feel free to explore the content.

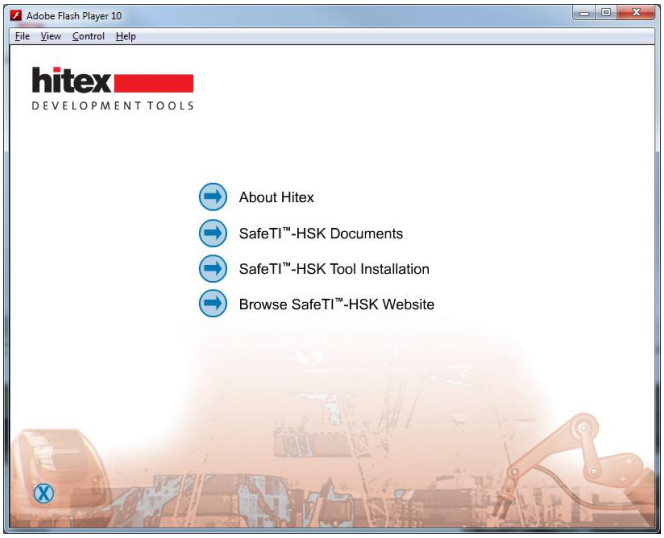

Figure 3-9 Tools DVD setup screen

Click on the arrow next to SafeTI™-HSK Tool Installation. Several tools from Texas Instruments are offered for installation. Refer to
Table **3-1** on the next page for further details.

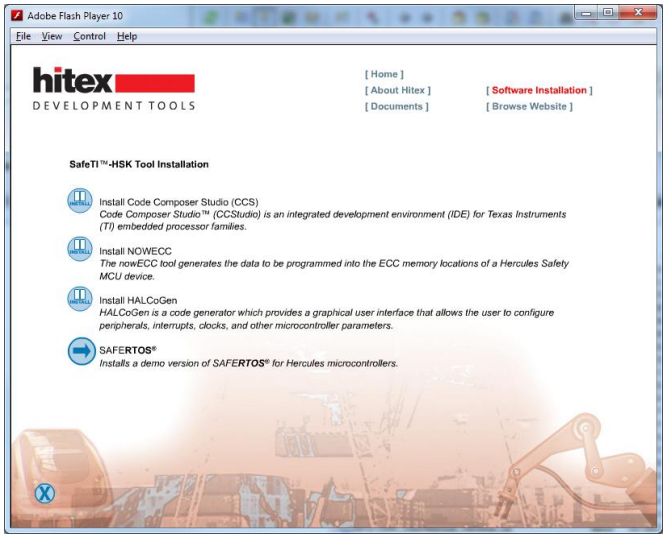
Adobe Flash Player 10
File View Control Help
hitex
DEVELOPMENT TOOLS
[Home]
[About Hitex]
[Documents]
[Software Installation]
[Browse Website]
SafeTI™-HSK Tool Installation
Install Code Composer Studio (CCS)
Code Composer Studio™ (CCStudio) is an integrated development environment (IDE) for Texas Instruments (TI) embedded processor families.
Install NOWECC
The nowECC tool generates the data to be programmed into the ECC memory locations of a Hercules Safety MCU device.
Install HALCoGen
HALCoGen is a code generator which provides a graphical user interface that allows the user to configure peripherals, interrupts, clocks, and other microcontroller parameters.
SAFERTOS®
Installs a demo version of SAFERTOS® for Hercules microcontrollers.

Figure 3-10 Tools DVD software installation

Tool		
Code Composer Studio	5.3.0.00090	Mandatory
NOWECC	V2.17	Mandatory
HALCoGen	03.05.00	Recommended
HET IDE		Optional
safeRTOS		Optional

Table 3-1 Provided Tools

For further information on these tools, please refer to their manuals and user's guides.

4 System description

The SafeTI™ HSK is a valuable evaluation tool to explore how to achieve functional safety with SafeTI™ Hercules microcontrollers and TPS6538x PMIC devices. This kit provides a hardware reference design and a software application framework to enable an application developer to build a safety application using TI's Hercules™ ARM® safety microcontrollers.

The following figure shows a block diagram of the SafeTI™ HSK.

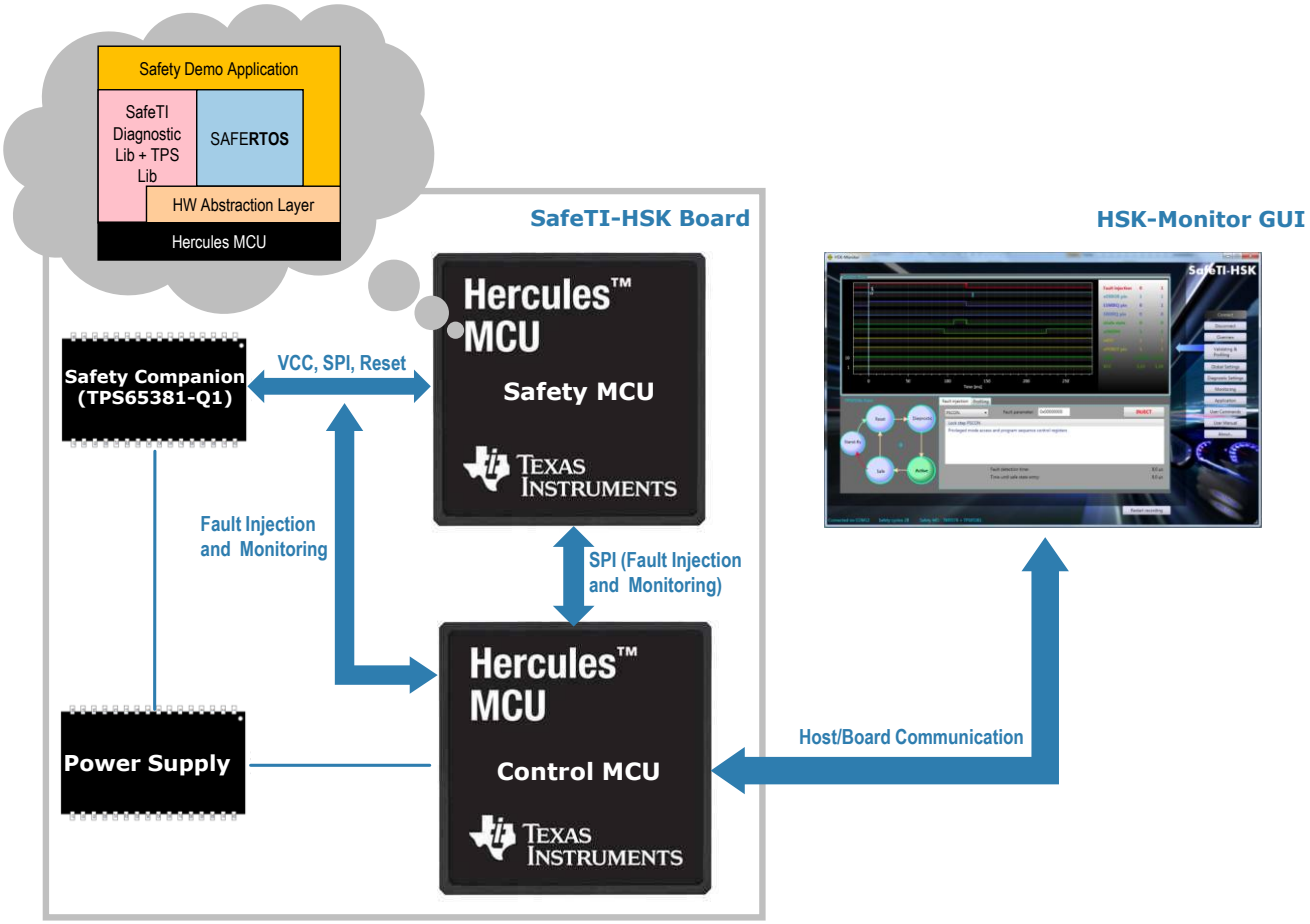

Figure 4-1: Main components of the kit

4.1 System components

The HSK hardware platform consists of:

- **Safety system**

The Safety system consists of the Safety MCU, the power supply companion chip, and a set of peripherals. These are used to demonstrate safety features of the two devices as would be used in an actual application.

  - **Safety MCU (SDUT):** The Safety MCU is one out of the different pin compatible variants of the Hercules™ ARM Safety Microcontrollers; TMS570LS3137 or RM48952. It interfaces with various other components on the board:
    - a multi-rail power supply with watchdog feature; the TPS65381.
    - a control and monitoring device, or the C&M device, which monitors the SDUT and TPS devices.
    - User peripherals: an accelerometer, a temperature sensor, an HMI (4xLED, potentiometer, LCD), a CAN transceiver, and a motor control interface (DIMM connector).

The HSK provides the ability to an end user to evaluate the safety features of this device, with the added ability to add other user functionality in terms of performance and connectivity.
An example *safety application* is provided in the HSK, which runs on the SDUT. This is an example application demonstrating key safety features of the Safety MCU and the power supply/watch dog companion chip TPS65381. The application is based on SafeRTOS and uses the SafeTI ™ Diagnostic Library. This software application provides the following user functionality:

- Read data from an accelerometer and a temperature sensor.
- Control an HMI, consisting of an onboard display as well as some LEDs and push buttons.
- Communicate with the TPS, an external watchdog via an SPI interface.
- Communicate with the C&M device for fault injection and monitoring.

Following sections in this chapter describe the sequence of operations in the safety application. Design of the safety application firmware is explained in chapter 10 of this manual in further detail.

- **Safety companion chip TPS65381**
  It is a Multi-Rail power supply controller for safety critical microcontrollers and provides the power supply to the Safety MCU. Additionally a watchdog is implemented and could be configured via SPI.

  In addition to these components, following features are available for evaluation of the SafeTI™-HSK with real world applications for analyzing their behavior in a safety system.

  - Accelerometer
  - Display
  - Potentiometer
  - Four push buttons
  - Four safety MCU controlled LEDs (blue)
  - One RGB LED
  - One LED (red) controlled by the C&M device
  - Temperature sensor

- **Control MCU (C&M)**

An RM48L952 microcontroller executes the “control and monitor” application. The main functions are:

- Communication with safety device over SPI.
- Injecting Faults to the safety device and monitor its behavior upon fault
- Supervision of TPS supply rails via inputs to an on chip ADC
- Communication with the HSK-Monitor GUI (via UART)
- Sample the signals asserted from the SDUT connected to GPIOs
- Exchange configuration data between the GUI and the SDUT.

- **HSK-Monitor GUI**

The safety features of the kit can be evaluated with the HSK-Monitor GUI. The user can trigger specific faults, which then will be injected to the safety device. Profiling measurements of the safety diagnostic features can be executed. Additionally, several states of the SDUT can be visualized in a task-monitor like view, together with the TPS operating states. The user has the capability to send 5 predefined user commands to the safety application software. For more information refer to the chapter “HSK-Monitor user manual”.

4.2 System behavior

4.2.1 *Startup, power on*

Upon power-on, the three components on the board start up: the C&M device, the TPS, and the SDUT. The flow is depicted in the figure below and detailed in the following sections.

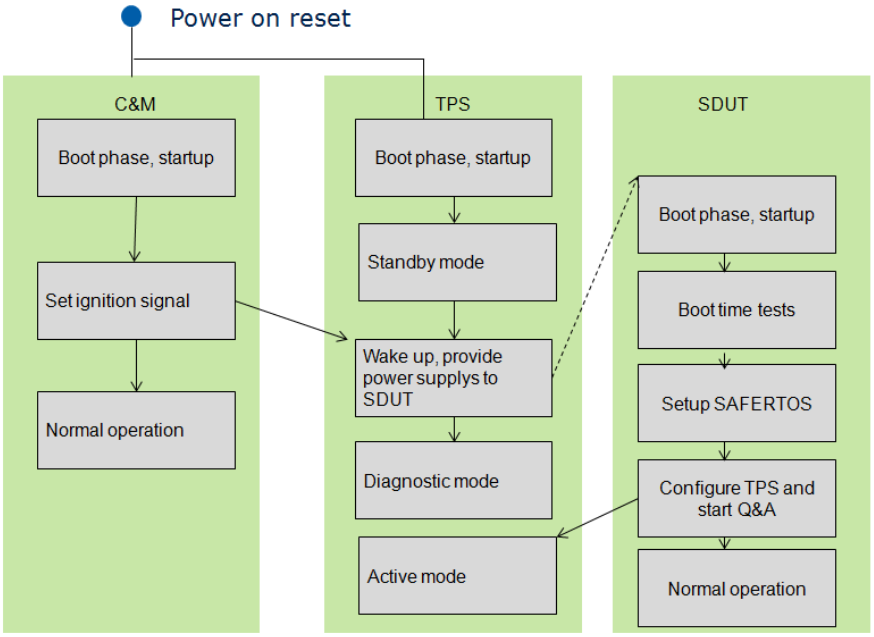

Figure 4-2: Startup Flow Diagram

4.2.1.1 Safety application firmware

The safety application firmware starts executing after the power on reset (nPORRST pin on the Hercules™ Safety MCU) is released by the TPS device. The software first executes tests called “boot time self-tests”. Following are the Self-tests executed at boot time:

- STC
- PBIST on RAM and FLASH
- Self-tests on TCRAM
- Self-tests FEE
- PBIST on Peripherals
- PSCON self-tests
- EFUSE
- CCM-R4
- ADC

Note: Configuration for setting up tests to run at boot-up is in the file “HSA_config.h”. The boot time code is in the sys_startup.c file.

In addition to the boot time self-tests, the controller peripherals are initialized in the startup code. The application firmware next creates all the task instances, queues and semaphores. Then the SafeRTOS scheduler is started, which takes control over the task execution in normal operation.

4.2.1.2 Control monitor device firmware

The control and monitor MCU starts executing its firmware after its reset is released – this is initiated through switching the power of the kit on. After initializing the stacks, memory and necessary peripherals, the *ignition* pin is asserted. The *ignition* pin is a GPIO output connected to the IGN pin of the TPS. This is important to consider, as the assertion of ignition wakes up the TPS device which in turn powers the safety system. The control monitor application then creates its tasks and queues and starts the scheduler. At this point, the boot phase is completed and normal operation is entered.

4.2.2 *External watchdog TPS*

The external watchdog controller, the TPS6538x, enters standby mode after the kit is powered on. It starts after receiving a wake up event triggered through the *IGN* input. A logical built in self test (BIST) is performed and upon success, the power signals to the SDUT are released. The TPS itself then moves to the “Diagnostic state” and waits to be initialized by the Safety application, which programs it over the SPI interface.

4.2.3 *Normal operation*

4.2.3.1 Safety application firmware

The main responsibilities are:

- Communicate with the C&M device
- Inject faults
- Operate the profile measurements
- Initialize the external watchdog device
- Service the TPS question and answer protocol
- Read sensor data (Accelerometer, Temperature sensor, push buttons)
- Print information to on-board display
- Assign the GPIO signals according to the firmware flow

These responsibilities are reflected in the software design of the Safety application in its task implementation and the inter-task communication channels. Note that there is one source and header file assigned to each task. The following is a list of the tasks along with a brief description of the tasks. For more detailed information, please refer to chapter 10.

Task instances

- **HSA_CMD_Handler (HSA_Command_Handler.c)**

This task handles the communication messages from and to the C&M device, such as user commands or application data. Activity on user LED1 indicates normal operation of this task.

- **HSA_FaultInjection (HSA_FaultInjection.c)**

This task handles the fault injection requests, like triggering a certain fault injection signal or calling the corresponding function out of the SDL.

- **HSA_SafetyLibServer (HSA_SafetyLibraryServer.c)**

This task handles two main jobs. The first is executing the cyclic run-time self-tests, which are activated in the HSK-Monitor GUI. The second job, which is executed after the first is completed, is to perform a profile measurement of a specific self-test, when activated by the user. Activity on user LED3 indicates normal operation of this task.

- **HSA_HMI_SensorAndHMI (HSA_SensorAndHMI.c)**

This task handles the data collection from the safety application’s peripherals. It is activated every 50ms. Activity on user LED2 indicates normal operation of this task.

- **HSA_DisplayTask (HSA_DisplayTask.c)**

This task handles the process of displaying information from other subtasks on the on-board display.

- **HSA_WDService_Task (HSA_WatchdogService.c)**

This task handles servicing the external watchdog within the time constraints set by the TPS window open/close time. To ensure fulfilling the time constraint, the SafeRTOS function TaskDelayUntil() is used. Activity on user LED4 indicates normal operation of this task.

- **HSA_Template_Task (HSA_TemplateTask.c)**

This task is intended as a template for the application developer to integrate own code.

5 Hardware Description

The SafeTI™-HSK serves as hardware and software platform to demonstrate safety featured applications using Texas Instruments' Hercules™ ARM® Safety MCUs. The kit allows developers to evaluate the safety features of the MCUs as well as to reuse these features to develop and execute their own safety software. The kit is presented in two variants depending on the safety MCU used: one with the TMS570LS3137 and one with the RM48L952.

5.1 Hitex Safety Kit Features

The SafeTI™-HSK comes with a rich set of on board features that facilitates the demonstration of safety applications. Key features include:

- A TI microcontroller (TMS570LS3137 or RM48L952) as a safety application MCU
- A second TI microcontroller (RM48L952) as a control and monitor MCU
- Two onboard USB XDS100v2 JTAG emulators (one for each MCU)
- ARM 20-pin JTAG debug header in 0.500 inch (1.27mm) pitch for external debugging of the safety MCU
- A multi-rail power supply with watchdog feature (TPS65381) for the safety MCU
- An onboard quad port USB hub
- SCI accessible through a USB virtual port (VCP)
- A CAN transceiver with screw terminal block
- One 128 x 32 pixels LCD module with white LED backlight (SPI Mode)
- Six user programmable LEDs (4 blue LEDs and 1 RGB LED connected to safety MCU; 1 red LED connected to C&M MCU)
- One red LED for the safety MCU and one red LED for the C&M MCU indicating reset states
- Four user programmable pushbuttons
- One reset pushbutton (nRST) connected to the safety MCU
- One digital accelerometer with SPI
- One ambient temperature sensor
- One 10 KOhm potentiometer
- TI DRV8312 controlCARD encoder and sensorless mode compatible interface
- Programmable onboard fault injection logic
- Power supply supporting 12V to 24V DC input

Figure 5-1 shows a block diagram of the main functional components on the kit. Items with dashed frames represent main functional components placed on the bottom side of the kit.

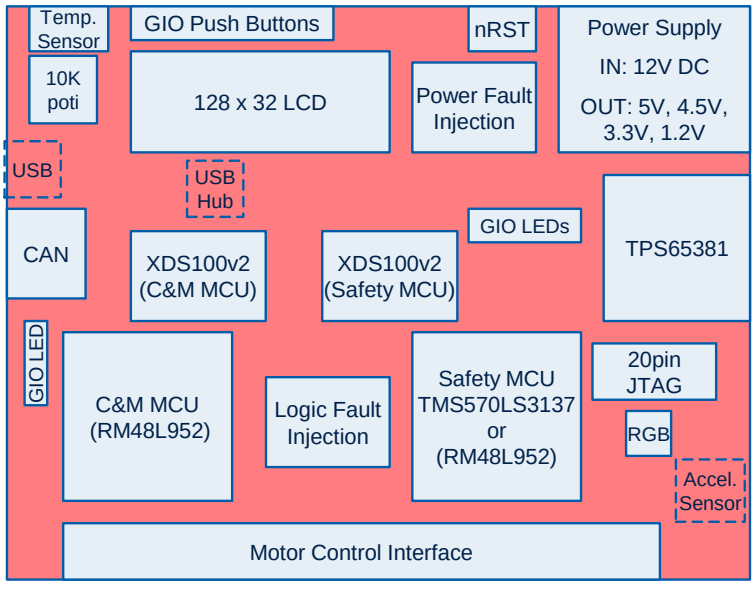

Figure 5-1 Block diagram of the kit

5.2 Physical Description

This section details the features of the SafeTI™-HSK board and their interfaces

5.2.1 *External Interfaces*

The SafeTI™-HSK board offers a number of interface ports used to connect the board to various external devices. These ports are listed below and detailed in the following sections.

Designator	Function
X300	ARM 20 pin 0.500 inch external JTAG connector
X500	TI DRV8312 controlCARD encoder and sensorless mode compatible interface
X501	CAN connector
X800	TPS65381 *DIAGOUT* pin jumper
X1000	Micro USB-AB connector
X1500	Power supply input connector

Table 5-1 External Interfaces

5.2.1.1 JTAG Connector

X300 offers a second debug channel to connect to the safety MCU via an external 3rd party JTAG emulator, used to debug ARM microcontrollers. The pinout of the connector is listed in the following table

Signal Name	Pin Number		Signal Name
Vref	1	2	N.C.
nTRST	3	4	GND
TDI	5	6	GND
TMS	7	8	Cable Detect (GND)
TCK	9	10	GND
RTCK	11	12	GND
TDO	13	14	GND
nRST	15	16	GND
N.C.	17	18	GND
N.C.	19	20	GND

Table 5-2 JTAG connector pin assignment

The SafeTI™-HSK features a cable detection circuit that senses when an external JTAG emulator is plugged onto **X300**. It then disables the onboard XDS100v2 emulator, and switches the indicator LED **D301** on.

5.2.1.2 TI DRV8312 controlCARD Interface

X500 offers an interface to connect the SafeTI™-HSK board to any TI board featuring the 100-pin DIMM connector. The signals brought out to the interface allow the integration of the SafeTI™-HSK in motor control applications, driven in encoder and sensorless compatible mode. The pinout of the interface is listed in the following table

Ball Number	Signal Name	Function	Pin Number		Function	Signal Name	Ball Number
	N.C.		1	51		N.C.	
	N.C.		2	52		N.C.	
	N.C.		3	53		N.C.	
	N.C.		4	54		N.C.	
	N.C.		5	55		N.C.	
	N.C.		6	56		N.C.	
	N.C.		7	57		N.C.	
	GND		8	58		GND	
U13	AD2IN[01]	IB-FB	9	59		N.C.	
	GND		10	60		GND	
U14	AD2IN[02]	VDCBUS	11	61		N.C.	
	GND		12	62		GND	
U16	AD2IN[03]	IA-FB	13	63		N.C.	
	GND		14	64		GND	
	N.C.		15	65		N.C.	
	N.C.		16	66		N.C.	
	N.C.		17	67		N.C.	
	N.C.		18	68		N.C.	
	N.C.		19	69		N.C.	
	N.C.		20	70		N.C.	
	N.C.		21	71		N.C.	
	N.C.		22	72		N.C.	
W3	N2HET1[06]	ePWM5A	23	73	ENDRV	GIOA[3]	E1
J1	N2HET1[18]	ePWM6A	24	74	ENDRV	GIOA[3]	E1
P2	N2HET1[20]	ePWM6B	25	75	ENDRV	GIOA[3]	E1
	N.C.		26	76		N.C.	
	GND		27	77		N.C.	
	N.C.		28	78		N.C.	
	N.C.		29	79		N.C.	
	N.C.		30	80		N.C.	
	N.C.		31	81		N.C.	
	N.C.		23	82		N.C.	
	N.C.		33	83	nTZ1	GIOA[4] N2HET1[29]	A6 C3
	N.C.		34	84	nTZ2	N2HET1[27] GIOA[5]	B2 B5
	N.C.		35	85		N.C.	
	N.C.		36	86		N.C.	
	GND		37	87		N.C.	
	N.C.		38	88		N.C.	

Ball Number	Signal Name	Function	Pin Number		Function	Signal Name	Ball Number
	N.C.		39	89		N.C.	
V2	N2HET1[01]	EQEP2A	40	90	EQEP2B	N2HET1[03]	U1
	N.C.		41	91	EQEP2I	N2HET1[04] GIOA[2]	B12 C1
	N.C.		42	92		N.C.	
	N.C.		43	93		N.C.	
	N.C.		44	94		N.C.	
	N.C.		45	95		N.C.	
	N.C.		46	96		N.C.	
	GND		47	97		N.C.	
	N.C.		48	98		N.C.	
	N.C.		49	99		N.C.	
	N.C.		50	100		N.C.	

Table 5-3 Motor control interface pin assignment

5.2.1.3 CAN Connector

X501 offers a high speed CAN communication link driven by CAN2 module of the safety MCU. The pinout of the CAN connector is listed in the following table

Pin Number	Function
1	CANH
2	GND
3	CANL

Table 5-4 CAN connector pin assignment

The CAN driver **U501** gets disabled when the TPS65381 deactivates the *ENDRV* signal.

5.2.1.4 DIAGOUT Jumper

The TPS65381 has the possibility to report diagnostic information through the *DIAGOUT* pin, which can be represented as analog measurement values as well as digital information. **X800** offers the possibility to connect this pin either to a digital input pin or to an analog input pin on the safety MCU as well as on the C&M MCU. By default the analog channel is jumpered. The following table lists the connection possibilities

Signal Name	Ball Number (Safety MCU)	Ball Number (C&M MCU)	Pin Number		Signal Name
A-DIAGOUT	W14	T19	1	2	DIAGOUT
D-DIAGOUT	A4	N1, G1	3	4	

Table 5-5 DIAGOUT jumper pin assignment

5.2.1.5 USB Connector

X1000 offers a communication link between the SafeTI™-HSK and a host PC. The USB port is connected to an on board USB hub, which manages the communication between the two on board XDS100v2 emulators and the host PC. This channel is used to debug the MCUs, to program them with their respective applications, and to communicate with demonstrator GUI.

5.2.1.6 Power Supply Connector

X1500 is the main power supply connector, which feeds in a supply voltage between +12V and +14V nominally into the board. The 2.5mm barrel type jack has the outer shell at negative potential and the inner pin as the positive potential, as shown in Figure 5-2.

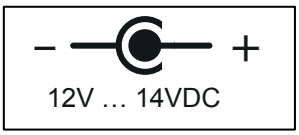

Figure 5-2 Input Power Supply Polarization

On board, the input +12V is fed to generate the +5V, +3.3V and +1.2V necessary to operate the USB debug interface, the onboard emulators, and the C&M functionality along with the fault injection block. Moreover, the +12V is fed through power switches **Q902** and **Q903** to power the TPS65381 at its inputs *VBAT* and *VBAT_SAFING* respectively, which in turn generates the +5V, +3.3V and +1.2V necessary for the safety MCU and its peripherals. Finally, the +12V is also used to generate a +4.5V power domain needed to simulate power faults onto the *VBAT* and *VBAT_SAFING* domains of the TPS65381.

5.2.2 *Display*

The SafeTI™-HSK features an on board 128 x 32 pixels LCD with white LED backlight. The display is connected to the safety MCU via SPI2 port. The following table lists the safety MCU signals used to drive the display

Display Signal	Safety MCU Signal	Functional Mode	Ball Number
CS1B	SPI2NCS[0]	SPI	N3
RST	nRST	Reset	B17
A0	SPI2NENA/SPI2NCS[1]	GPIO	D3
SCL	SPI2CLK	SPI	E2
SDA (SI)	SPI2SIMO	SPI	D1

Table 5-6 Display interface

5.2.3 *LEDs*

The SafeTI™-HSK features a number of LEDs used as static indicators, or are available for the application. The following table lists all the static LEDs along with their designated functions

Designator	Color	Function
D300	Red	Safety MCU Reset (nRST)
D301	Blue	External JTAG Emulator present
D600	Red	C&M MCU Reset (nRST)
D904	Red / Green	Status indication of VBAT Green: 12V present (nominal supply) Red: 4.5V present (undervoltage)
D905	Red / Green	Status indication of VBAT_SAFING Green: 12V present (nominal supply) Red: 4.5V present (undervoltage)
D1100	Blue	Safety XDS100v2 SCI RX
D1101	Blue	Safety XDS100v2 SCI TX

Designator	Color	Function
D1202	Blue	Safety XDS100V2 PWRENn
D1300	Blue	C&M XDS100v2 SCI RX
D1301	Blue	C&M XDS100v2 SCI TX
D1402	Blue	C&M XDS100V2 PWRENn
D1500	Blue	Power present (12V nominal)
D1502	Blue	5V present

Table 5-7 LED indicators

The following table lists the application programmable LEDs, along with their drivers

Designator	Color	Driver	Signal	Ball Number
D501	Blue	Safety MCU	GIOB[4]	G1
D502	Red / Green / Blue	Safety MCU	N2HET1[0]: Green N2HET1[28]: Blue N2HET1[31]: Red	K18 K19 J17
D503	Blue	Safety MCU	GIOB[5]	G2
D504	Blue	Safety MCU	GIOB[6]	J2
D505	Blue	Safety MCU	GIOB[7]	F1
D700	Red	C&M MCU	SPI2ENA/SPI2NCS[1]	D3

Table 5-8 User-programmable LEDs

5.2.4 *Push Buttons*

The SafeTI™-HSK features five push buttons connected to the safety MCU, four of which are application programmable. The last one is connected to the *nRST* signal to trigger a warm reset. The following table lists the push buttons along with their designated ports

Designator	Receiver	Signal	Ball Number
SW300	Safety MCU	nRST	B17
SW500	Safety MCU	GIOB[0]	M2
SW501	Safety MCU	GIOB[1]	K2
SW502	Safety MCU	GIOB[2]	F2
SW503	Safety MCU	GIOB[3]	W10

Table 5-9 User-programmable push buttons

5.2.5 *Sensors*

The SafeTI™-HSK features a number of sensors that may be used by the safety application. The following sections details these sensors along with the connections to the safety MCU

5.2.5.1 Digital Accelerometer

The SafeTI™-HSK features a small, thin, ultralow power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ±16 g. The digital output data is formatted as 16-bit twos complement and is accessible to the safety MCU through a 4-wire SPI digital interface. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (3.9

mg/LSB) enables measurement of inclination changes less than 1.0°. Activity is signaled to the safety MCU through two extra interrupt pins. The following table lists the signals between the safety MCU signals and the sensor

Sensor Signal	Safety MCU Signal	Functional Mode	Ball Number
$\overline{CS}$	MIBSPI5NCS[0]	SPI	E19
SDO/ALT ADR	MIBSPI5SOMI[0]	SPI	J18
SDA/SDI/SDIO	MIBSPI5SIMO[0]	SPI	J19
SCL/SCLK	MIBSPI5CLK	SPI	H19
INT1	GIOA[0]	GPIO	A5
INT2	GIOA[1]	GPIO	C2

Table 5-10 Digital accelerometer interface

5.2.5.2 Temperature Sensor

The SafeTI™-HSK features a simple 100KOhm NTC thermistor as an ambient temperature sensor. The thermistor is connected in a voltage divider circuit manner along with a 100KOhm resistor to an analog input, from which the safety MCU can then derive the ambient temperature. The following table lists the thermistor along with its connection to the safety MCU

Designator	Receiver	Signal	Ball Number
R507	Safety MCU	AD1IN[01]	V17

Table 5-11 Temperature sensor interface

5.2.5.3 Potentiometer

The SafeTI™-HSK features a 10KOhm potentiometer that delivers to the safety application any voltage between 0V and 5V. The following table lists the potentiometer along with its connection to the safety MCU

Designator	Receiver	Signal	Ball Number
R503	Safety MCU	AD1IN[02]	V18

Table 5-12 Potentiometer interface

5.2.6 *Test Points*

The SafeTI™-HSK features a set of test points for probing signals relevant to the safety application. Figure 5-3 points out the respective locations of these test points.
The test points are divided into two groups depending on their “from – to” connections. The first group is for signals between the safety MCU and the TPS65381, and the second is for signals between the safety MCU and the C&M MCU. The table below lists the test points between the safety MCU and the TPS65381.

Designator	Net Name	From	Signal	Ball / Pin Number	To	Signal	Ball / Pin Number
TP01	S-nPORST	TPS65381	nRES	6	Safety MCU	nPORST	W7
TP02	WD-Error	Safety MCU	nERROR	B14	TPS65381	ERROR / WDI	13
TP03	S-GIOA3	TPS65381	ENDRV	32	Safety MCU	GIOA[3] / N2HET2[2]	E1
TP04	WD_SCK	Safety MCU	MIBSPI3 CLK	V9	TPS65381	SCLK	11

Designator	Net Name	From	Signal	Ball / Pin Number	To	Signal	Ball / Pin Number
TP05	WD_NCS	Safety MCU	MIBSPI3 NCS[0]	V10	TPS65381	NCS	8
TP06	WD_SDI	Safety MCU	MIBSPI3 SIMO	W8	TPS65381	SDI	9
TP07	S3-SOMI	TPS65381	SDO	10	Safety MCU	MIBSPI3 SOMI	V8
TP08	VCCAD	TPS65381	VDD5	20	Safety MCU	VCCAD	W15
TP09	VCCIO	TPS65381	VDD3/5	21	Safety MCU	VCCIO VCCP	F6 – F8, F11 – F14, G6, G14, H6, H14, J6, L14, M6, M14, N6, N14, P6 – P9, P12 – P14
TP10	VCC	TPS65381	VT_1V2		Safety MCU	VCC VCCPLL	F9, F10 H10, J14 K6, K8, K12, K14 L6, M10, P10, P11

Table 5-13 Test points between Safety MCU and TPS65381

The following table lists the test points between the safety MCU and the C&M MCU.

Designator	Net Name	From	Signal	Ball / Pin Number	To	Signal	Ball / Pin Number
TP11	C1-SCK	C&M MCU	MIBSPI1 CLK	F18	Safety MCU	MIBSPI1 CLK	F18
TP12	C1-NCS0	C&M MCU	MIBSPI1 NCS[0]	R2	Safety MCU	MIBSPI1 NCS[0]	R2
TP13	C1-SIMO	C&M MCU	MIBSPI1 SIMO	F19	Safety MCU	MIBSPI1 SIMO	F19
TP14	C1-SOMI	Safety MCU	MIBSPI1 SOMI	G18	C&M MCU	MIBSPI1 SOMI	G18
TP15	C-IRQ	C&M MCU	MIBSPI1 NCS[2]	G3	Safety MCU	GIOA[7]	M1
TP16	S-ERRINJ	Safety MCU	RTP_ DATA[13]	C4	C&M MCU	N2HET1[29] GIOB[5]	A3 G2
TP17	S-ESMIRQ	Safety MCU	RTP_ DATA[12]	C5	C&M MCU	N2HET1[30] GIOB[6]	B11 J2
TP18	S-ERRIRQ	Safety MCU	RTP_ DATA[11]	C6	C&M MCU	N2HET1[31] GIOB[7]	J17 F1
TP19	S-GPIO0	Safety MCU	RTP_ DATA[06]	C10	C&M MCU	GIOA[0]	A5
TP20	S-GPIO1	Safety MCU	RTP_ DATA[05]	C11	C&M MCU	GIOA[1]	C2
TP21	S-GPIO2	Safety MCU	RTP_ DATA[04]	C12	C&M MCU	GIOA[2]	C1
TP22	S-GPIO3	Safety MCU	RTP_ DATA[03]	C13	C&M MCU	GIOA[3]	E1
TP23	S-GPIO4	Safety MCU	RTP_ DATA[01]	C14	C&M MCU	GIOA[4]	A6

Designator	Net Name	From	Signal	Ball / Pin Number	To	Signal	Ball / Pin Number
TP24	S-GPIO5	Safety MCU	RTP_nENA	C15	C&M MCU	GIOA[5]	B5
TP25	S-GPIO6	Safety MCU	RTP_nSYNC	C16	C&M MCU	GIOA[6]	H3
TP26	S-GPIO7	Safety MCU	RTP_CLK	C17	C&M MCU	GIOA[7]	M1

Table 5-14 Test points between Safety MCU and C&M MCU

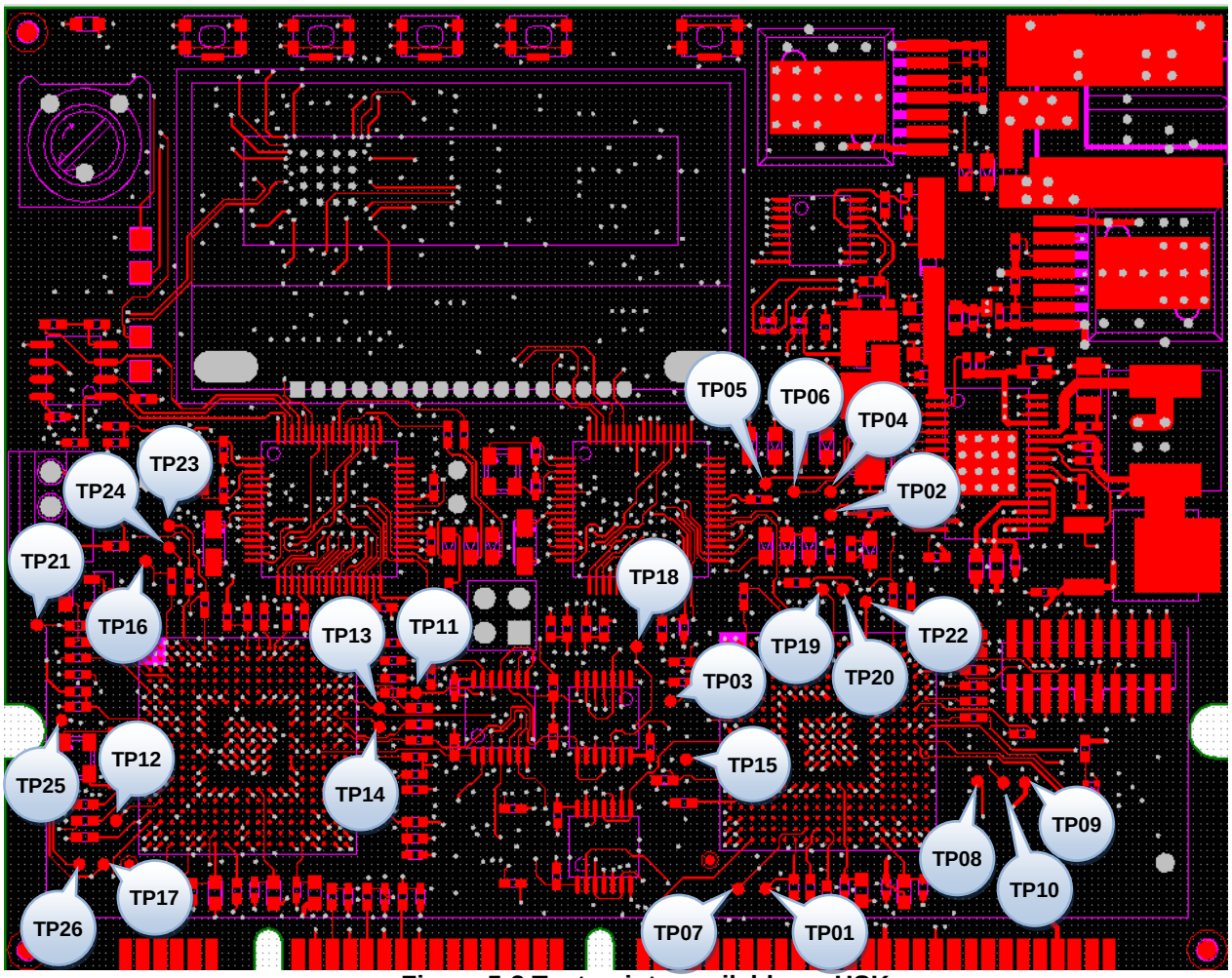

Figure 5-3 Test points available on HSK

Some of these signals are captured and displayed directly by the HSK-Monitor GUI, to demonstrate the features of the kit. For detailed information about the GUI, please refer to chapter 6. The following table maps the test point signals captured to the signals displayed on the GUI.

Designator	Net Name	GUI Signal Name
TP01	S-nPORST	nPORST
TP02	WD-Error	nESMERR pin
TP03	S-GIOA3	ENDRV
TP03	S-GIOA3 inverted	Safe State TPS
TP16	S-ERRINJ	Fault Injection
TP17	S-ESMIRQ	ESMIRQ pin
TP18	S-ERRIRQ	ERR Detection
TP19 – TP22	S-GPIO0 – S-GPIO3	*used for task monitoring (0-15)*
TP23	S-GPIO4	Periodic tests
TP25	S-GPIO6	Safe State MCU
TP26	S-GPIO7	*used for profiling*

Table 5-15 Mapping between test points and GUI signals

6 HSK Monitor Graphical User Interface (GUI)

6.1 Main Window

The GUI that comes with the SafeTI™-HSK is started by executing the HSK-Monitor.exe file. The Main Window after start-up is shown in Figure 6-1. The Main Window contains the Main Viewing Area (1), the Navigation Bar (2), hyperlinks (3), the button “Stop recording” (4) and the Status Bar (5). All elements are further described in the following subsections.

Figure 6-1 GUI Main Window after start-up (edited)

6.1.1 *Main Viewing Area (1)*

The Main Viewing Area is used to display what we refer to as GUI pages (see Section 6.2). The displayed content depends on the function selected from the Navigation Bar (2). See also Section 6.1.2.

6.1.2 *Navigation Bar (2)*

The Navigation Bar consists of 11 buttons. These buttons provide the menu interface to the user to select functionality of the HSK. The functions of these buttons are described in Table 6-1. Enabled buttons are shown in light gray, disabled buttons are shown in dark gray and cannot be selected.

Navigation Bar Button	Description
Connect	The evaluation board communicates with the PC over USB. A Virtual COM port driver causes the board to appear as an additional COM port available to the PC. In order to connect the board to the GUI, the corresponding COM port (e.g. COM33←HSK) has to be selected. After the connection is established, the Overview Page is shown in the Main Viewing Area (see Figure 6-2).
Disconnect	Disconnects the connection between the PC GUI application and the board.
Overview	Displays the Overview Page within the Main Viewing Area (see also Section 6.2.1).
Validation & Profiling	Displays the Validation & Profiling Page within the Main Viewing Area (see also Section 6.2.2).
Global Settings	Displays the Global Settings Page within the Main Viewing Area (see also Section 6.2.3).
Diagnostic Settings	Displays the Diagnostic Settings Page within the Main Viewing Area (see also Section 6.2.4).
Monitoring	Displays the Monitoring Page within the Main Viewing Area (see also Section 6.2.5).
Application	Displays the Application Page within the Main Viewing Area (see also Section 6.2.6).
User Commands	Displays the User Commands Page within the Main Viewing Area (see also Section 6.2.6).
User Manual	Opens the user manual with a pdf reader.
About...	Shows copyright information as well as the monitor version, the application version, the board revision, the board type, the GUI version and the GUI date in an additional window. A hyperlink that leads to an update website is also given.

Table 6-1 Buttons in the Navigation Bar

A selected button is indicated by a blue arrow pointing to the current content of the Main Viewing Area (Figure 6-2).

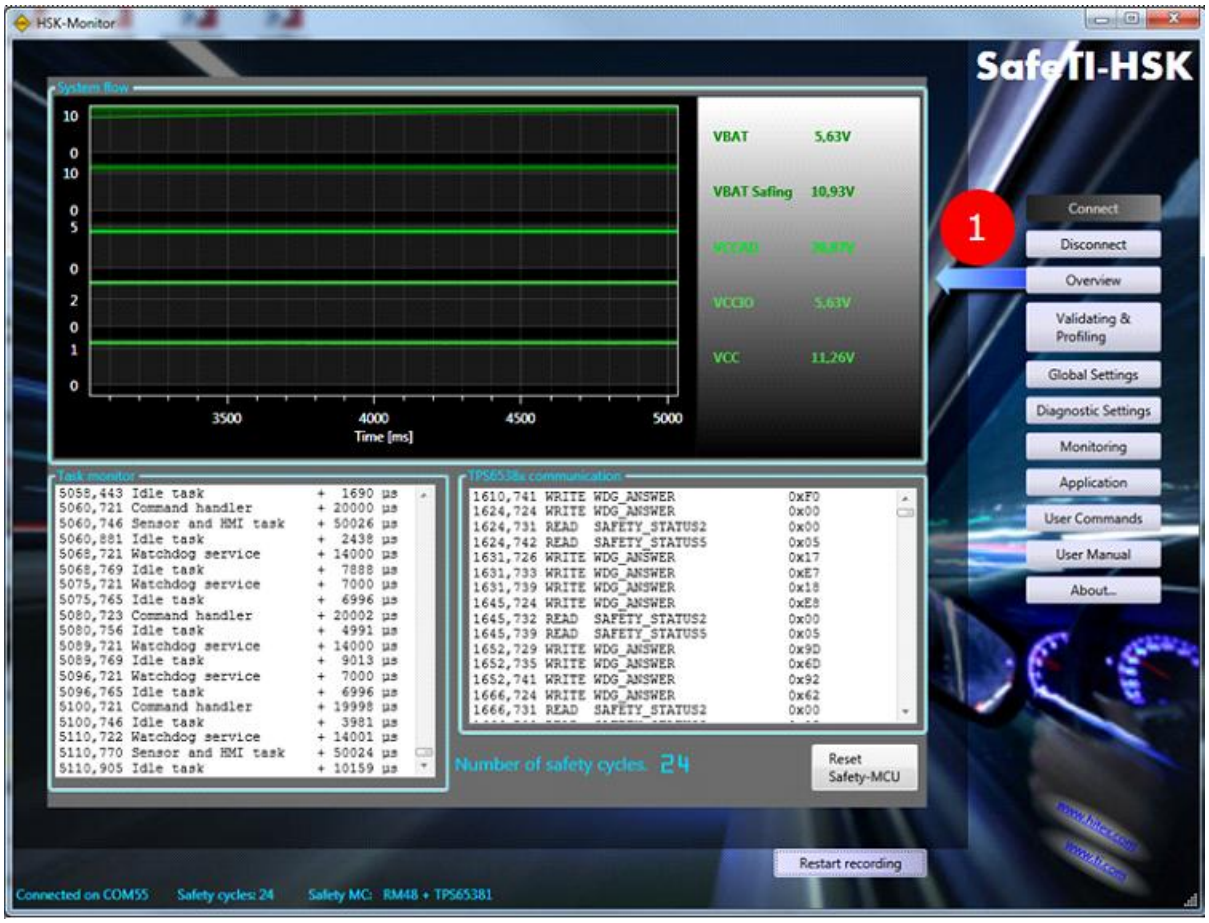

Figure 6-2 GUI Main Window after connection

6.1.3 *Hyperlinks (3)*

The hyperlinks lead to the Hitex and Texas Instruments websites.

6.1.4 *Button "Stop recording" (4)*

With the button "Stop recording", the recording of events and data for the GUI pages is stopped (e.g. to allow an in-depth examination of the GUI content). When recording stops, the buttons name changes to "Restart recording". When the button is pressed again, the recording of events and data is restarted after resetting the SDUT and the GUI Pages (except the pages used for configuration).

6.1.5 *Status Bar (5)*

The Status Bar gives information about:

- The connection of the board and the GUI
- The number of safety cycles executed by the safety microcontroller after reset (a single execution of all microcontroller tests is counted as one safety cycle)
- The type of the microcontroller and the companion chip assembled on the connected board.

6.2 GUI Pages

The GUI has seven pages. These pages are selected via the corresponding buttons within the Navigation Bar (see also
Table 6-1) and displayed in the Main Viewing Area. The structure and functionality of the pages are described in the Sections 6.2.1 to 6.2.7.

6.2.1 *Overview Page*

The Overview Page gives a general overview of the activities and the supply voltages within the system. It is depicted in Figure 6-3. The page consists of three main parts, the System flow (1), the Task monitor (2) and the TPS6538x communication monitor (3).

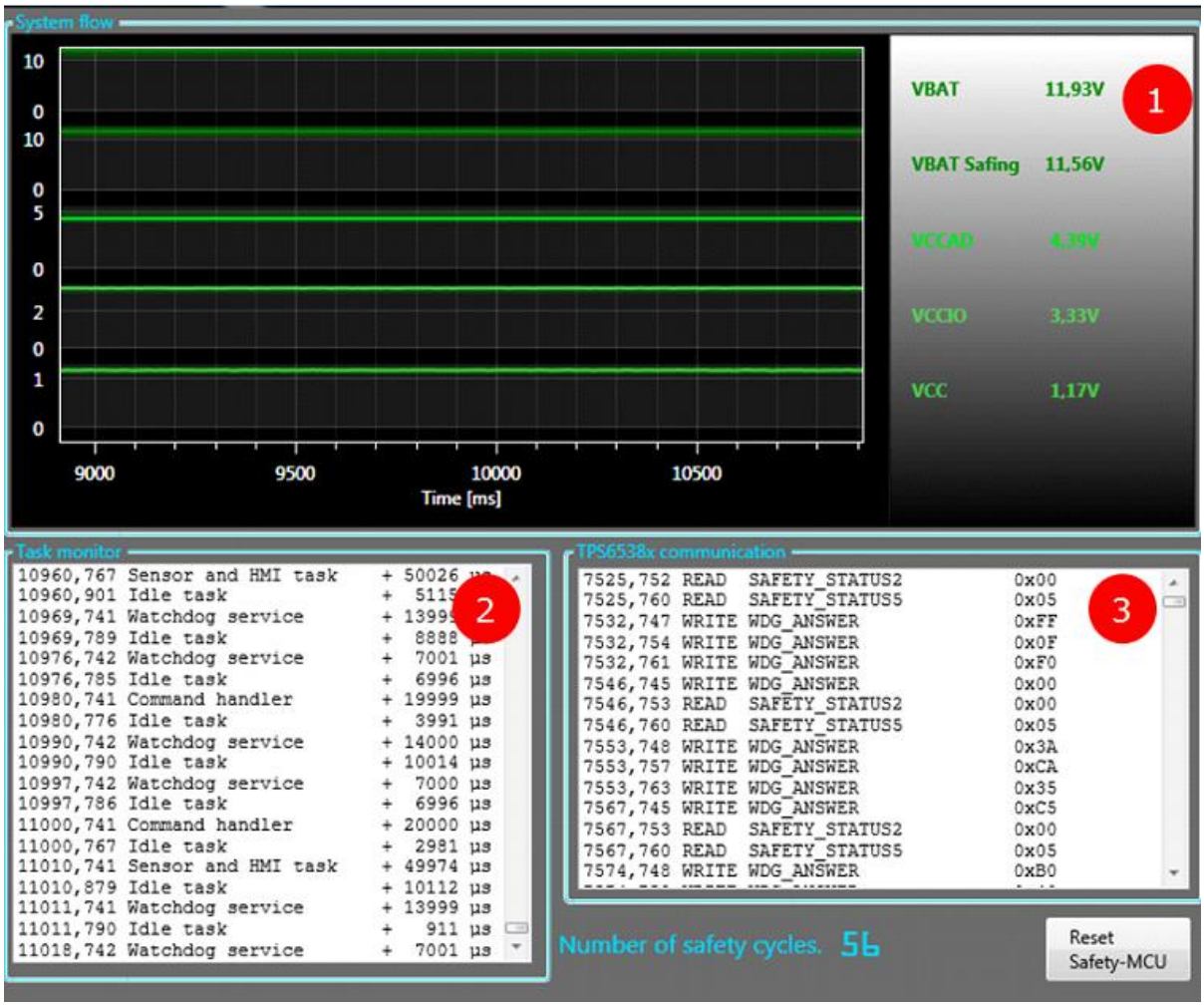

Figure 6-3 Overview Page

The System flow (1) depicts voltages to and from the TPS6538x over time. The voltages are compiled in

Table **6-2**.

Voltage	Description
VBAT	Power supply for the TPS
VBAT Safing	Power supply for the supervision modules in the TPS
VCCAD	Supply voltage for Safety MCU analog digital converter cores (provided by TPS)
VCCIO	Safety MCU IO supply voltage (provided by TPS)
VCC	Safety MCU core supply voltage (provided by TPS)

Table 6-2 Information within the System flow part of the Overview Page

Zooming of information shown in the System flow part of the Overview page can be done best using a mouse wheel. A click with the left mouse button onto a signal to gives timing and value information. A context window for more detailed information can be opened with a right mouse click. The signals over time can be horizontally moved by clicking and holding the right mouse button within the System flow and a subsequent movement of the mouse.

The Task monitor (2) gives information about the task execution on the safety MCU. More specifically, for each task the start time (in ms), the name and the time that has elapsed since the last execution is given (in us). The latter should correspond to the cycle time during normal operation.

The TPS6538x communication monitor (3) shows the recording of the SPI communication between the safety MCU (master) and the TPS6538x companion (slave). Each line in the TPS6538x communication monitor corresponds to one SPI frame which consists of an 8 bit SPI command phase and an 8 bit SPI data phase. In the first column, time stamps for the SPI frames are given. The second and the third column give information regarding the SPI command phase – whether a register within the TPS6538x is read or written and the name of the corresponding register. The fourth column gives the data value that is transmitted during the SPI data phase.

6.2.2 *Validation & Profiling Page*

The Validation & Profiling Page is for fault injection and profiling measurements. It is depicted in Figure 6-4 and consists of three parts, the System flow (1), the TPS6538x state machine (2) and the Fault injection and Profiling control part (3).

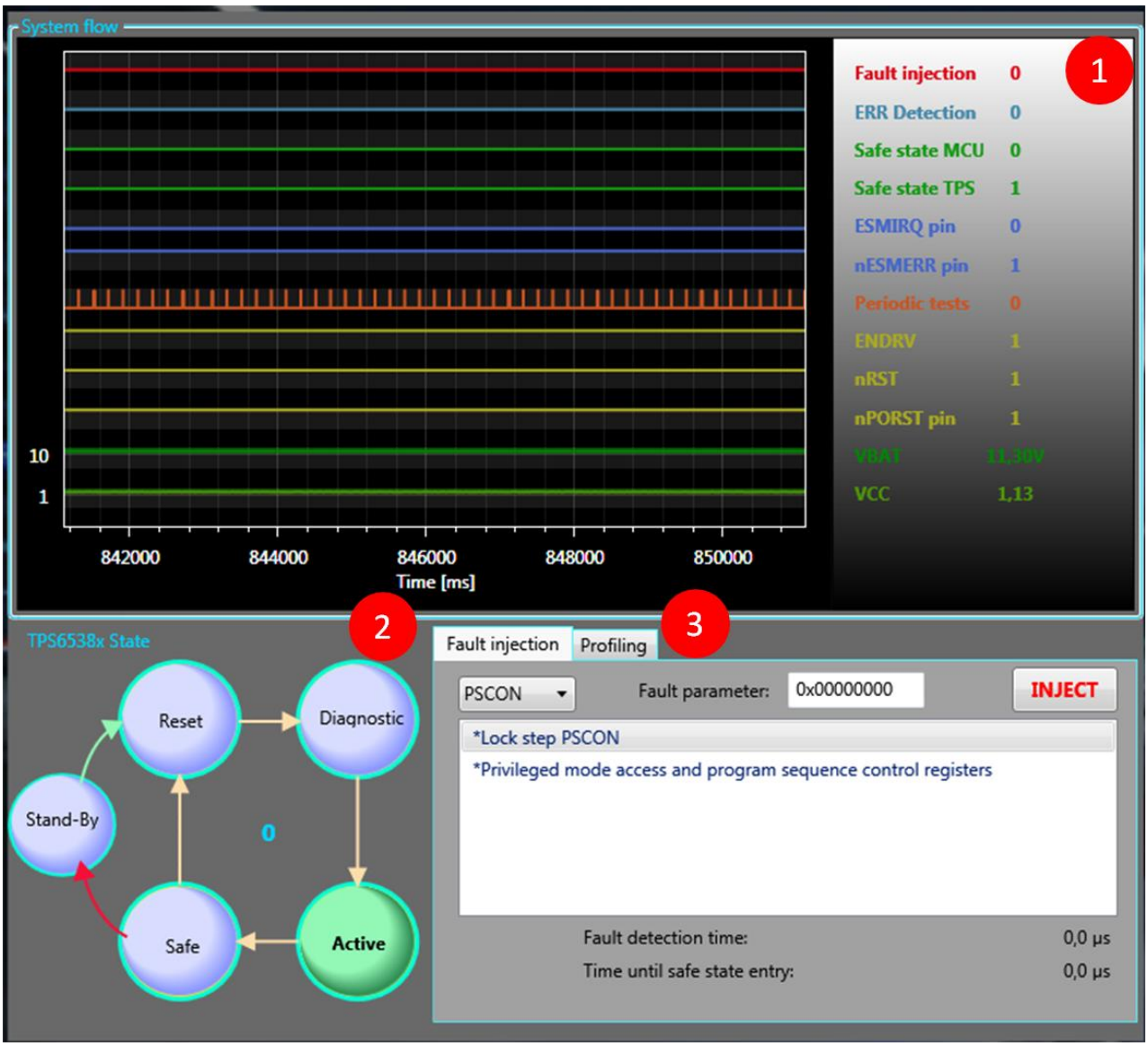

Figure 6-4 Validation & Profiling Page

The content of the System flow part is determined by the tab selection within the Fault injection and Profiling control part (either Fault injection or Profiling).
The signals that are displayed when the Fault injection tab is selected are compiled in Table 6-3. The handling is similar to the handling of the System flow part in the Overview Page (see Section 6.2.1).

Signal	Description	Meaning
Fault injection	GPIO signal raised by either application before a fault is generated.	Fault appears
ERR Detection	GPIO signal set when an error is detected by the safety application firmware. An error is detected when 1. an abort handler is called 2. the ENDRV is detected low 3. an ESM interrupt is detected	Fault detected
Safe state MCU	Signal indicates that the MCU is in safe state. The MCU is regarded in safe State when PORST is active low.	Safe state reached

Signal	Description	Meaning
Safe state TPS	Signal indicates that a safe state is reached. Despite to the “Safe state CPU” signal here the external watchdog recognized the error in the system and drives it to safe state.	Safe state reached
ESMIRQ pin	GPIO signal set in ESM interrupt handler.	Fault has been detected by ESM
Periodic tests	A GPIO signal that indicates the execution start of the periodic run time tests.	Run time test execution
nESMERR pin	Signal issued by the ESM of the safety MCU.	Fault indication to external devices
ENDRV	Output signal of the TPS to indicate that the error counter has exceeded its limit.	Safe state reached
nRST	Signal of the safety MCU to indicate a reset state.	System restarts
nPORST pin	Connected to nRES output of the TPS.	Power down safety MCU
VBAT	Supply voltage for the TPS	
VCC	Supply voltage for the safety MCU core (provided by TPS).	

Table 6-3 Signals within the System flow window part during fault injection

When the Profiling tab is selected in the Fault injection and Profiling part, the content of the System flow part just shows the signal that is used for profiling purposes (see also Chapter 7).

The TPS6538x state shows the current operating state of the companion chip. The current state is highlighted with a light green filling of the corresponding state. Further information on the companion chip and its states can be found in (Texas Instruments, 2012).

The Fault injection window in the Fault injection and Profiling control part can be used to inject a fault into the running system (see also Chapter 7). With a drop-down menu (see (1) in Figure 6-5) the unit, in which the fault shall be injected, is selected. The faults available for injection for the corresponding unit are then displayed in a sub-window arranged below (2). A fault for injection can be selected by a mouse click. If the selected fault injection shall be configured, a parameter value can be specified (3). However, this feature is currently not used. The actual fault injection is triggered when the “INJECT” button is pressed (4). The fault detection time and the time until a safe state is reached are displayed at the bottom of the Fault injection window (5).

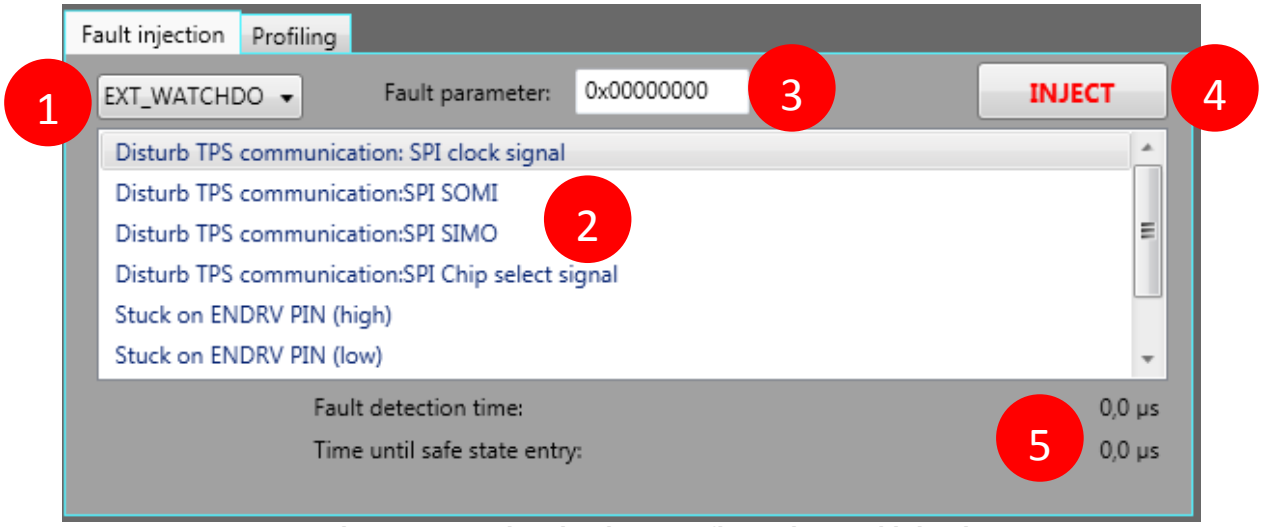

Figure 6-5 Fault selection, configuration and injection

The Profiling window in the Fault injection and Profiling control part can be used to conduct timing measurements for certain runtime test (7). With a drop-down menu (see (1) in Figure 6-6), the unit for which the test is to be performed is selected. The available tests for the corresponding unit are then displayed in a sub window arranged below (2). A test can be selected by a mouse click. The measurement is started when the “Profiling” button is pressed (3). As alternative to a single test, a full set of tests can be executed and measured by clicking on the “Profiling full safety task” button (see (4) in Figure 6-6). The set of tests can be defined through the Diagnostic Settings page, which is explained in 6.2.4.The measured profiling time is shown at the bottom in either case (5).

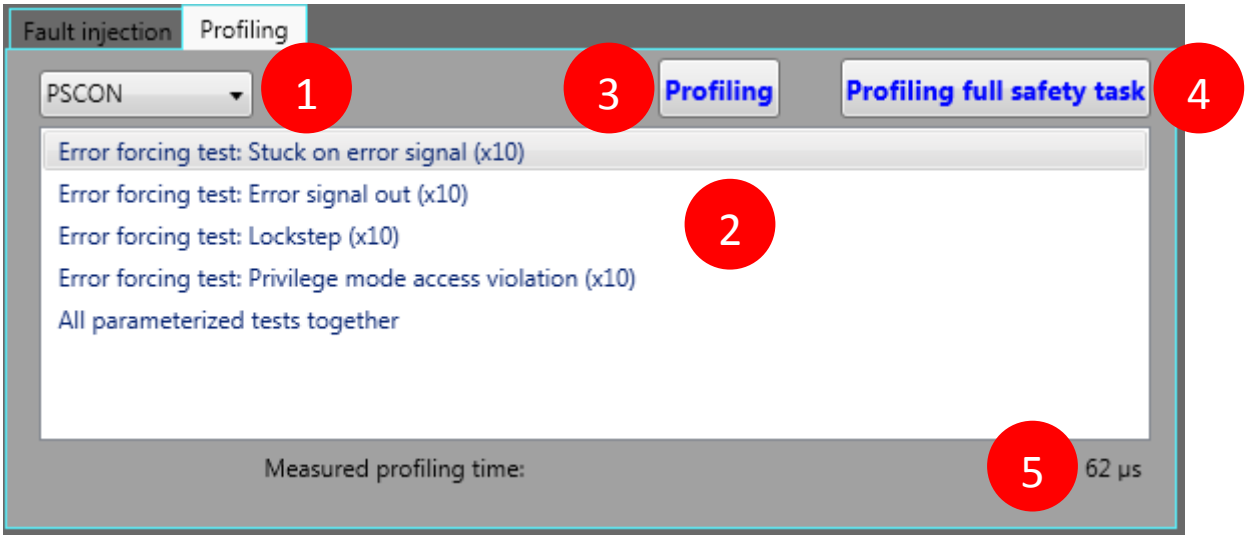

Figure 6-6 Test profiling in the Validation & Profiling Page

6.2.3 *Global Settings Page*

The overall behavior of the system can be influenced by some settings, which can be changed within the Global Settings Page. The page is depicted in Figure 6-7. It consists of four parts, a Safety loop slider (1), a System load slider (2), ESM Settings (3) and TPS6538x Settings (4). It is possible to save and load settings (5). Clicking the corresponding buttons opens a file menu. The current settings are applied to the running system when the “Apply” button is clicked (6). The four parts of the Global Settings Page are described in the following subsections.

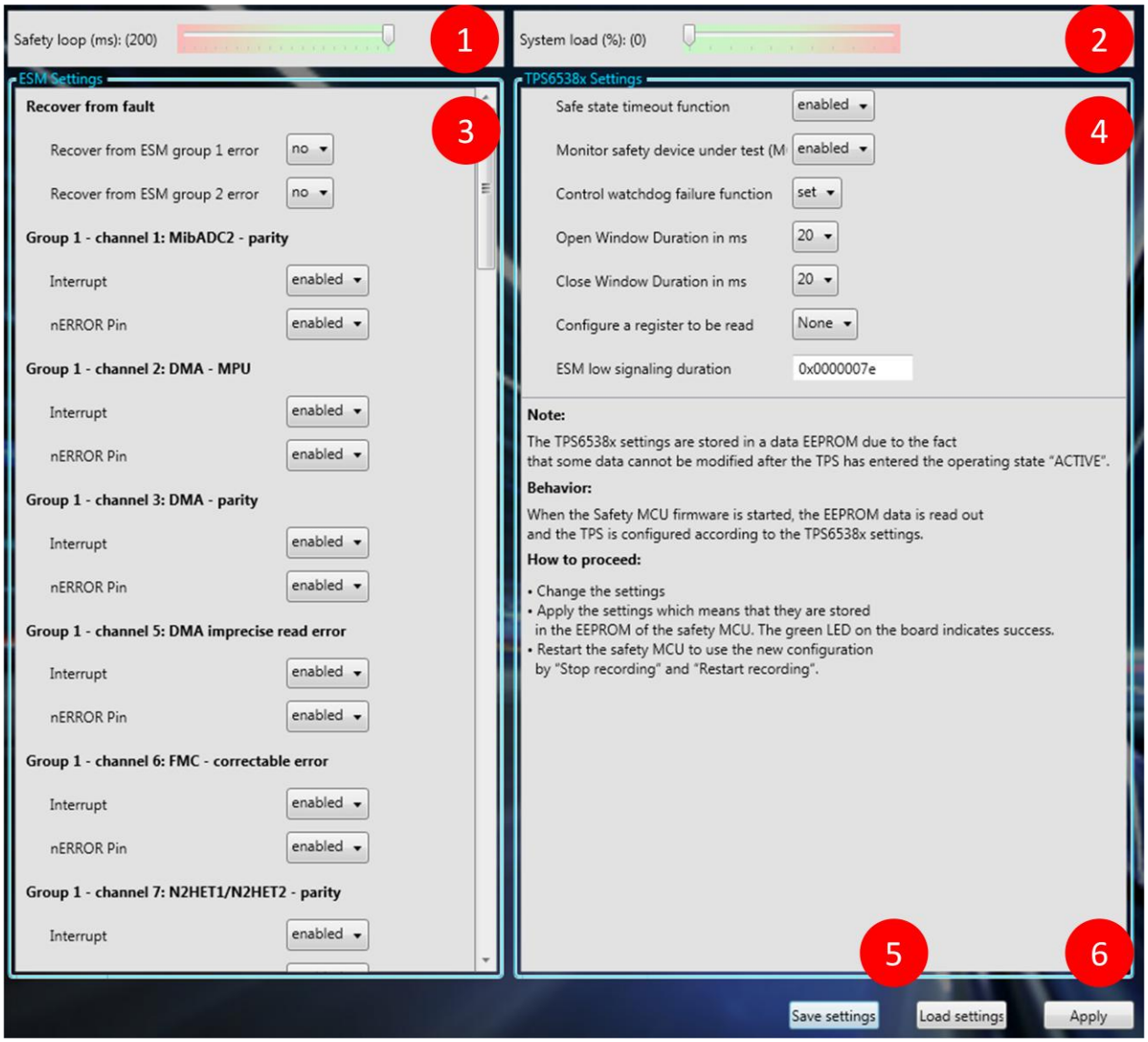

Figure 6-7 Global Settings Page

6.2.3.1 Safety loop

With the safety loop slider the cycle time of the run time tests can be changed. The application task which triggers the run time tests splits the execution of these tests into 5 slots. In each slot a specific subset of the tests is executed. After the execution of all 5 slots all tests have been processed. If, for example, the safety loop slider has a value of 100 ms, this means that all tests have to be processed within 100ms. It is obvious that a small time interval for the tests increases the system load.

6.2.3.2 System load

The application executed on the safety MCU could be stressed with a task that just wastes processor time (see also Chapter 10). It is implemented in the RTI compare interrupt handler, which is configured to produce an interrupt each 200 us. In this interrupt handler, a loop is processed according to the system load parameter. The task execution can be set with the system load slider in the Global Settings Page. If the system load selected is too high, the watchdog service task is prevented from running in time. Also the profiling times depend on the system load.

6.2.3.3 ESM Settings

The Error Signaling Module (ESM) is a hardware module in the safety MCU. It is responsible to control the error signal output of the Safety MCU (cf. nESMERR signal in Table 6-3). The errors which may occur are partitioned into three groups. Only group one errors can be configured to produce an interrupt and/or set the error signal. Group two and three errors always raise the error signal output pin. The group one error channels that can be configured over drop-down menus in the ESM Settings part of the Global Settings Page are compiled in Table 6-4.

There are two other parameters which are not applicable in the ESM module itself instead they control the behavior of the safety application firmware when an ESM interrupt occurs. For ESM group1 and group 2 errors, it is possible to configure if the system recovers from the fault or not. This is specified using the parameters:

- Recover from group1 error (yes/no)
- Recover from group2 error (yes/no)

If a recover from a group error is set to “yes” then the firmware in the ESM interrupt handler routine resets the fault and continues with normal operation. If the Value is “no” then the firmware in the ESM interrupt handler calls “safestate” routine which enters an endless loop. As a consequence the external watchdog (TPS Q&A) is no more serviced.

Channel	Error
1	MibADC2 - parity
2	DMA - MPU
3	DMA - parity
5	DMA imprecise read error
6	FMC - correctable error
7	N2HET1/N2HET2 - parity
8	HET TU1/HET TU2 - parity
9	HET TU1/HET TU2 - MPU
10	PLL slip
11	Clock Monitor interrupt
13	DMA - imprecise write error
15	VIM RAM parity
17	MibSPI1 parity
18	MibSPI3 parity
19	MibADC1 parity
21	DCAN1 - parity
22	DCAN3 - parity
23	DCAN2 - parity
24	MibSPI5 - parity
26	B0TCM correctable error
27	CPU self-test
28	B1TCM correctable error
30	DCC1 error
31	CCMR4 self-test

Channel	Error
35	FMC - correctable error
36	FMC - uncorrectable error
37	IOMM - Mux configuration error
38	PSCON compare error
39	PSCON self-test error
40	EFuse controller error
41	EFuse self-test error
42	PLL2 - Slip
43	Ethernet Controller master interface
62	DCC2 - error

Table 6-4: ESM error configuration

6.2.3.4 TPS6538x Settings

The TPS setting that can be changed within the Global Settings Page are compiled in Table 6-5, with the respective register name or bit-field indicated under the setting name.

Setting	Description
Safe state timeout function *SAFETY_FUNC_CFG[7]*	Controls the SAFE state time-out function. If enabled, the device transitions to the RESET state 680 ms after the error counter has exceeded its limit. If disabled, the device remains in the SAFE state when the error counter has exceeded its limit.
Monitor safety device under test *SAFETY_CHECK_CTRL[2]*	Controls the MCU_ERROR pin function. In our case the MCU_ERROR input pin is connected to the nESMERROR pin of the safety device. If enabled, the MCU_ERROR pin failure is monitored and detected. If disabled, the MCU_ERROR pin failure is not monitored.
Control watchdog failure function *SAFETY_FUNC_CFG[3]*	When set, a watchdog failure is detected when the watchdog failure counter reaches a value of 7. This leads to a transition from ACTIVE to RESET state. When cleared, the device remains in current state when the watchdog failure counter reaches a value of 7.
Open Window Duration *WDT_WIN1_CFG*	Open time window duration; (value +1) x 0.55) ms
Close Window Duration *WDT_WIN2_CFG[4:0]*	Close time window duration; (value +1) x 0.55) ms

Setting	Description
Low signaling duration (ESM) *SAFETY_ERR_PWM_L*	The TPS features a monitor which supervises the external ERROR input pin from MCU. An MCU signaling error condition is detected when the ERROR pin remains low for a programmed amount of time set by the SAFETY_ERR_PWM_L register (the low signaling duration).
Configure a register to be read	If a register is selected, this register is cyclically read out. The value can be checked in the TPS6538x communication monitor in the Overview Page (see (3) in Figure 6-3).

Table 6-5 TPS Settings

6.2.4 *Diagnostic Settings Page*

The execution of the periodic runtime tests and the parameterized profiling tests can be configured by the settings made in the Diagnostic Settings Page. The page is depicted in Figure 6-8. It is divided in two parts, the Safety Diagnostics (1) and the Safety Diagnostics Settings (2). It is possible to save and load settings (3). Clicking the corresponding buttons opens a file menu. The current settings are applied to the running system when the “Apply” button is clicked (4).

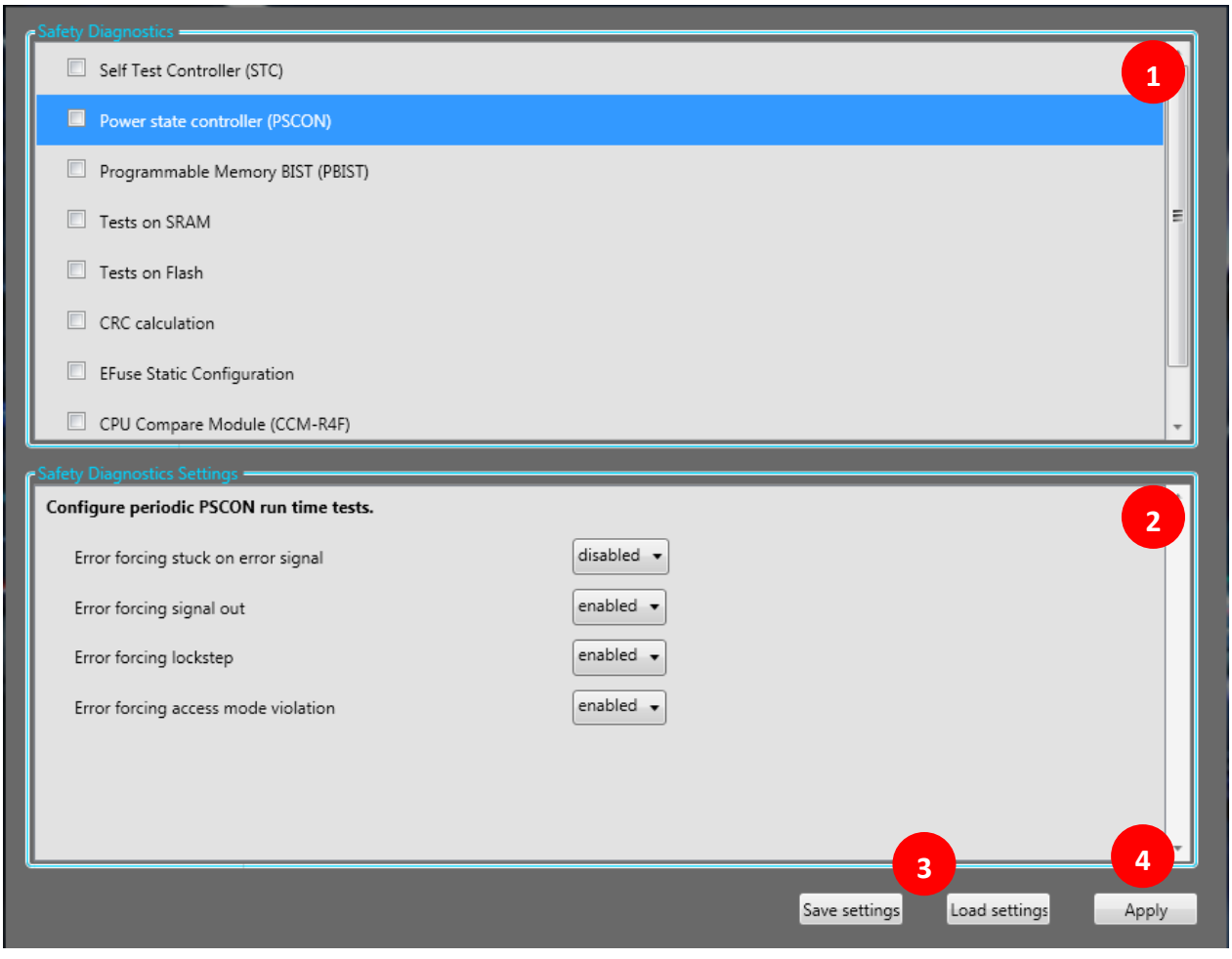

Figure 6-8 Diagnostic Settings Page

By marking the checkboxes besides the *test groups* in the Safety Diagnostics, the tests that are conducted periodically at runtime can be set.
Selecting a *test group* will show the settings for that *test group* in the Safety Diagnostic Settings part of this page. The selection is highlighted with a blue bar and can be changed by clicking on a different group. In Figure 6-8, the Power state controller group is selected and the corresponding settings are shown. The settings can be adapted by the drop-down menus in the Safety Diagnostic Settings. The changes made here impact the cyclic runtime tests as well as the parameterized profiling tests.
The groups and corresponding settings are further detailed in the following subsections.

6.2.4.1 Group - Self Test Controller (STC)

- Logic built-in self-test:
  - enable: STC test is executed at run time
  - disable: STC test is not executed
- Interval Count: a Value of 1 up to 24 is allowed. This specifies the amount of intervals which are operated with one STC test run.
- Run timeout: The total number of VBUS clock cycles it will take before a self-test timeout error (TIMEOUT_ERR) will be triggered after the initiation of the self-test run. This is a fail-safe feature to not hang-up the system on account of any run away self-test issues.

6.2.4.2 Group - Power state controller (PSCON)

- Error forcing stuck on error signal
  - enable: PSCON stuck in error signal test is executed at run time
  - disable: The test is disabled
- Error forcing signal out
  - enable: PSCON error forcing signal out test is executed at run time
  - disable: The test is disabled
- Error forcing lockstep
  - enable: Error forcing lockstep test is executed at run time
  - disable: The test is disabled
- Error forcing access mode violation
  - enable: Error forcing access mode violation test is executed at run time
  - disable: The test is disabled

6.2.4.3 Group - Programmable Memory BIST (pBIST)

- PBIST GROUP: The selection of the RAM group which shall be used for a test.
- Algorithm: Selection of the algorithm which is used to test the memory.
- Memory type: Selection of the memory type (Single Port, Two Port, ROM)
- Store/Restore selected RAM: It can be configured if the concerned RAM space shall be saved previously to the test and restored after the test execution or not.

Note: Since not all memories can be tested with each algorithm care should be taken that the enabled memory groups can be tested with that algorithm. For further information on that please take a look into the technical reference manual of the microcontroller device.

6.2.4.4 Group - Tests on SRAM

Several tests can be enabled / disabled for run time execution.

- Error forcing 1 Bit
- Error forcing 2 Bit
- Address and control parity
- Redundant address decode

Furthermore, with “DataECC” the data ECC logic can be switched on/off for the RAM.

6.2.4.5 Group - Tests on Flash

Several tests can be enabled / disabled for run time execution.

- Error forcing 1 Bit
- Error forcing 2 Bit

Furthermore, with “DataECC” the data ECC logic can be switched on/off for the Flash memory.

6.2.4.6 Group – CRC calculation

The CRC value of a specific memory area can be calculated and checked with the value calculated at boot time. Two parameters provide two checks at run time.

- CRC on flash code range
- CRC on the VIM RAM

Two additional parameters provide the possibility to define a memory range for a CRC calculation. These parameters are intended for the profiling measurement.

- Start address
- End address

6.2.4.7 Group – Efuse Static Configuration

- Autoload self-test
  - enable: Autoload self-test is executed at run time
  - disable: The test is disabled
- EFuse self-test ECC
  - enable: ECC test on Efuse is executed at run time
  - disable: The test is disabled
- EFuse self-test stuck at zero
  - enable: Stuck at zero test is executed at run time
  - disable: The test is disabled

6.2.4.8 Group – CPU compare module (CCM-R4F)

- Self-test (Lockstep)
  - enable: The CPU lockstep test is executed at run time
  - disable: The test is disabled
- Error forcing test
  - enable: The CPU error forcing test is executed at run time
  - disable: The test is disabled
- Self-test error forcing
  - enable: Error forcing self-test is executed at run time
  - disable: The test is disabled

6.2.4.9 Group – ADC

The ADC can be tested. With the parameter the “ADC”, a channel can be selected to be tested.

- ADC1
- ADC2

6.2.5 *Monitoring Page*

The Monitoring Page is depicted in Figure 6-9. It is divided into three parts, the Voltages ((1), see also Section 6.2.1), the Version information ((2), see also the “About…” button in the Navigation Bar) and the Application messages (3).

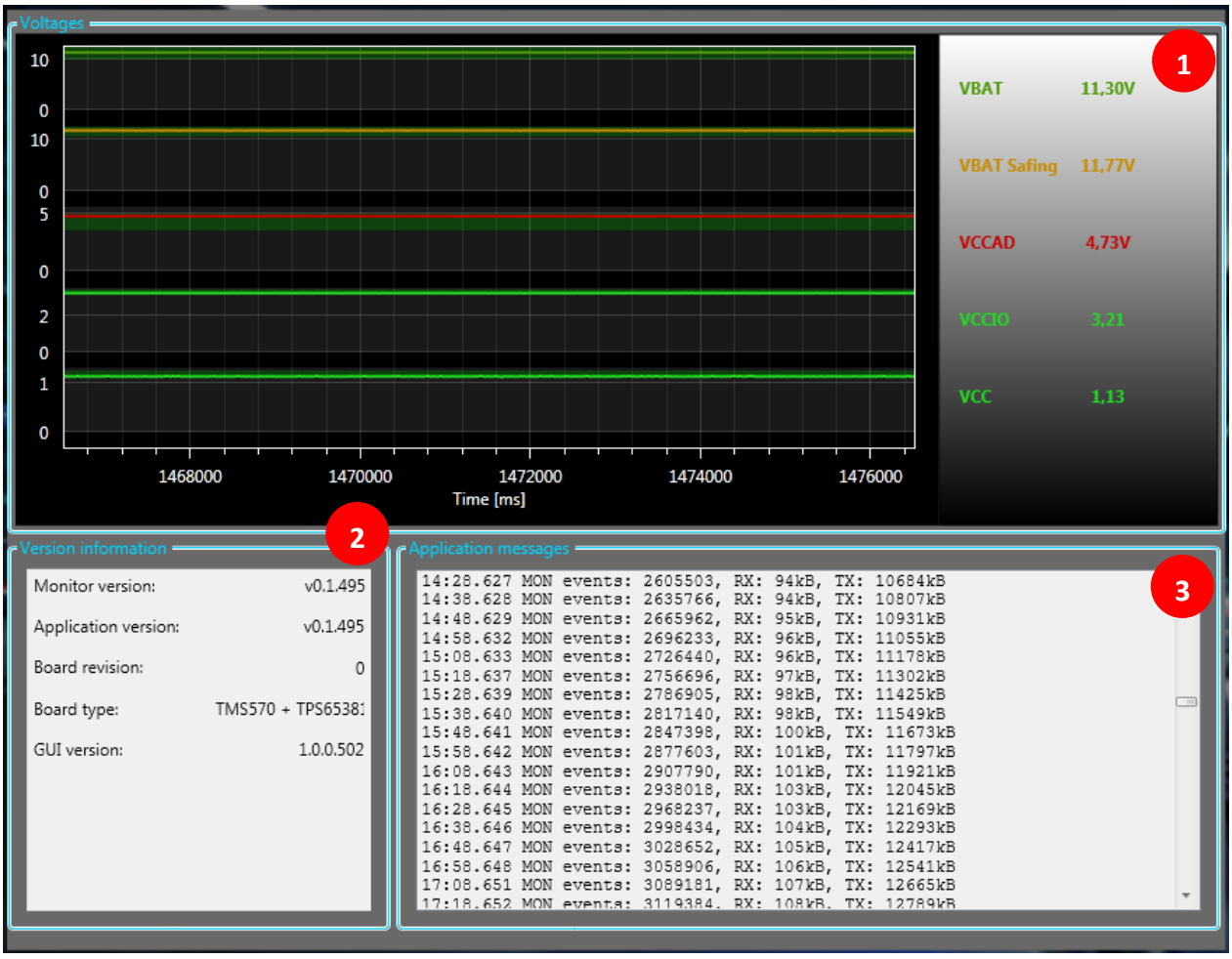

Figure 6-9 Monitoring Page

The application messages part of the Monitoring Page acts as a terminal to output user defined strings tagged with timestamps from the applications running on either the safety MCU or the C&M device. The messages can be used for debugging purposes.

6.2.6 *Application Page*

The Application Page depicted in Figure 6-10, shows temperature and acceleration values. There are 3 acceleration axes displayed ‘X’ (horizontal), ‘Y’ (horizontal orthogonal to ‘X’) and ‘Z’ vertical. The values are standardized to acceleration with respect to gravity. The application executed on the safety MCU reads the corresponding sensor data and subsequently performs certain conversion calculations. Finally, the results are transferred to the GUI.

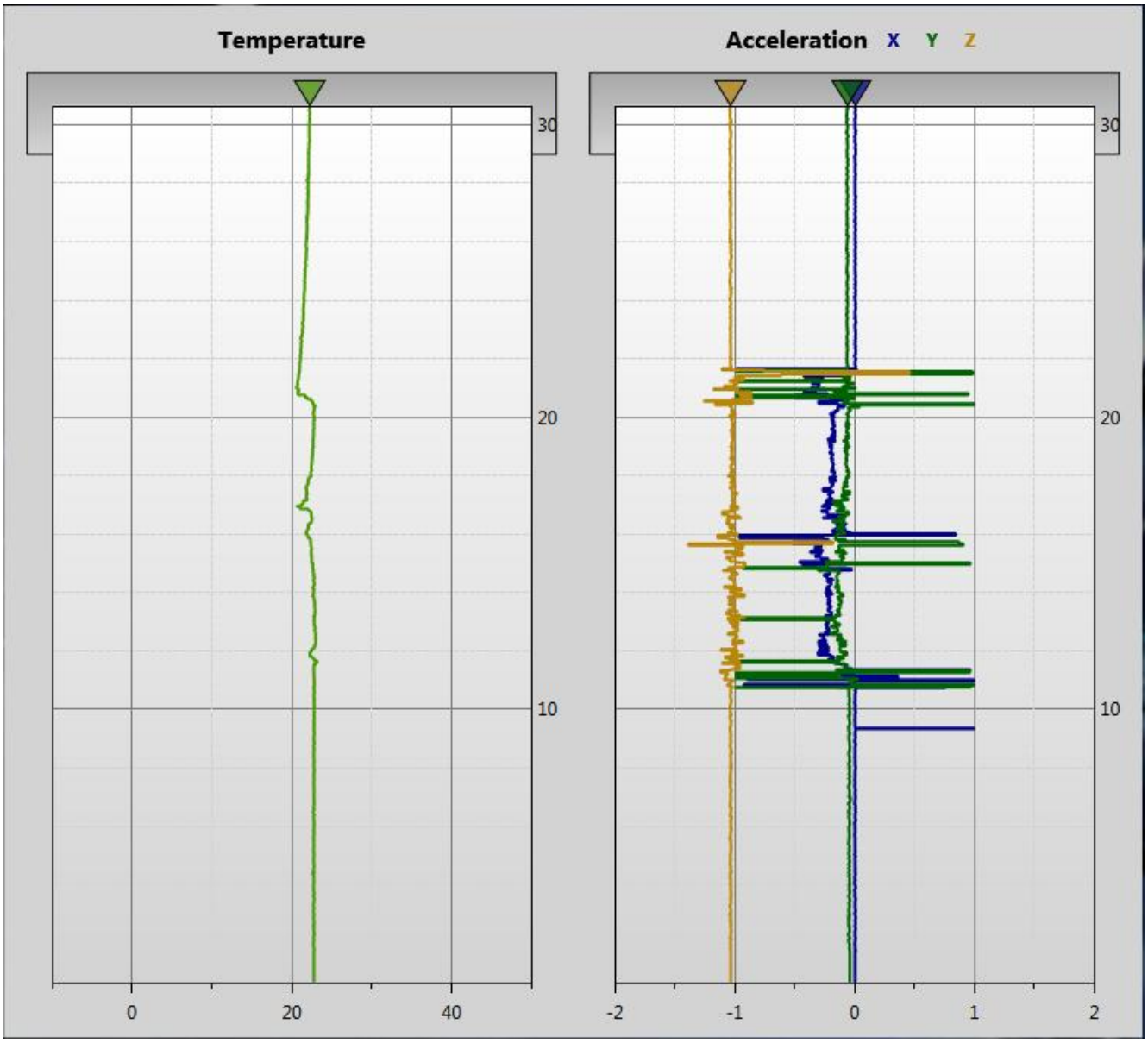

Figure 6-10 Application Page

6.2.7 *User Commands Page*

The User Commands Page is depicted in Figure 6-11. It consists of two parts, the User commands (1) and the Application messages ((2), see also the Application messages part in the Monitoring Page, Section 6.2.5).
The safety MCU application contains a task that is provided for user extensions. The User commands part of the User Commands Page is a convenient way for users to send arguments from the GUI to this task. By clicking the “Execute” button, the corresponding argument becomes available to the task upon its next execution.

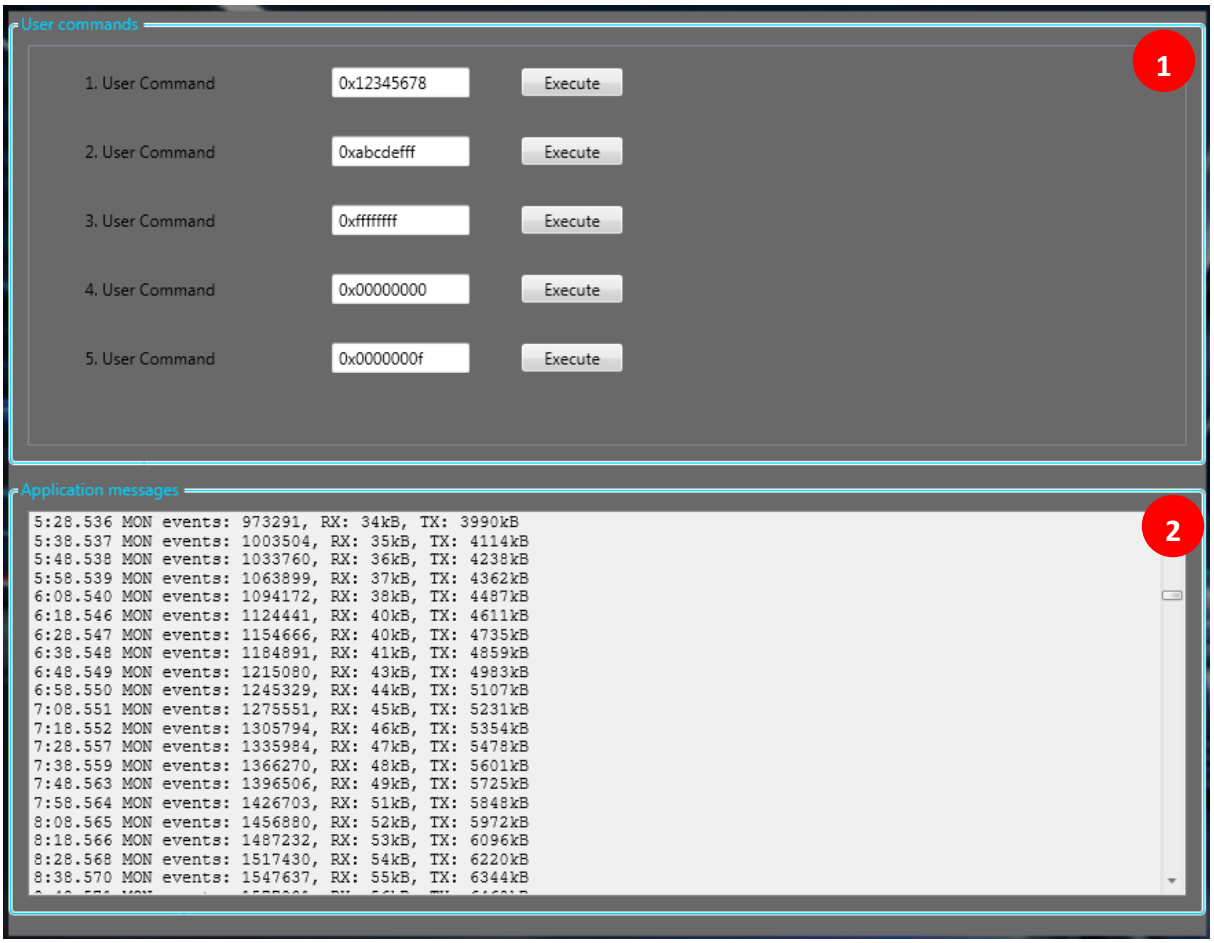
User commands
1
1. User Command
0x12345678
Execute
2. User Command
0xabcdefff
Execute
3. User Command
0xffffffff
Execute
4. User Command
0x00000000
Execute
5. User Command
0x0000000f
Execute
Application messages
2
5:28.536 MON events: 973291, RX: 34kB, TX: 3990kB
5:38.537 MON events: 1003504, RX: 35kB, TX: 4114kB
5:48.538 MON events: 1033760, RX: 36kB, TX: 4238kB
5:58.539 MON events: 1063899, RX: 37kB, TX: 4362kB
6:08.540 MON events: 1094172, RX: 38kB, TX: 4487kB
6:18.546 MON events: 1124441, RX: 40kB, TX: 4611kB
6:28.547 MON events: 1154666, RX: 40kB, TX: 4735kB
6:38.548 MON events: 1184891, RX: 41kB, TX: 4859kB
6:48.549 MON events: 1215080, RX: 43kB, TX: 4983kB
6:58.550 MON events: 1245329, RX: 44kB, TX: 5107kB
7:08.551 MON events: 1275551, RX: 45kB, TX: 5231kB
7:18.552 MON events: 1305794, RX: 46kB, TX: 5354kB
7:28.557 MON events: 1335984, RX: 47kB, TX: 5478kB
7:38.559 MON events: 1366270, RX: 48kB, TX: 5601kB
7:48.563 MON events: 1396506, RX: 49kB, TX: 5725kB
7:58.564 MON events: 1426703, RX: 51kB, TX: 5848kB
8:08.565 MON events: 1456880, RX: 52kB, TX: 5972kB
8:18.566 MON events: 1487232, RX: 53kB, TX: 6096kB
8:28.568 MON events: 1517430, RX: 54kB, TX: 6220kB
8:38.570 MON events: 1547637, RX: 55kB, TX: 6344kB

Figure 6-11 User Commands Page

7 Fault injection

The intention of the fault injection feature is to evaluate the behavior of the example safety system when faults are injected into it. These faults can be injected with the GUI and then disturb the normal application flow. With the “Diagnostic Settings” and the “Global Settings” it is possible to change the example safety system behavior.
Select the “Validating & Profiling” page. The “fault injection” tab is preselected. The various faults are grouped for a better handling. A fault is injected when the button “INJECT” is clicked.

7.1 Data flow

If a fault is injected using the GUI a command with a unique ID related to the selected fault is sent via the GUI to the C&M device. The C&M device investigates the fault ID and decides if it is a fault which has to be produced by itself or by the SDUT. If the fault has to be produced by the safety application the command is forwarded.
Before the fault is generated the “Fault injection signal” is raised.
The C&M device samples all the monitored signals with timestamps and sends the information to the GUI. After a short time duration the sampling stops and the GUI display is frozen to give the user a chance to evaluate what has been sampled. To go on with the next fault injection it is required to restart the recording. Note that with “Restart recording” the safety device is restarted.
Since the data flow (and as a consequence the signal flow) depends on various parameter settings and the fault which is injected, a categorization has been provided below.

7.2 System behavior, categorization

7.2.1 *Signals and their meaning*

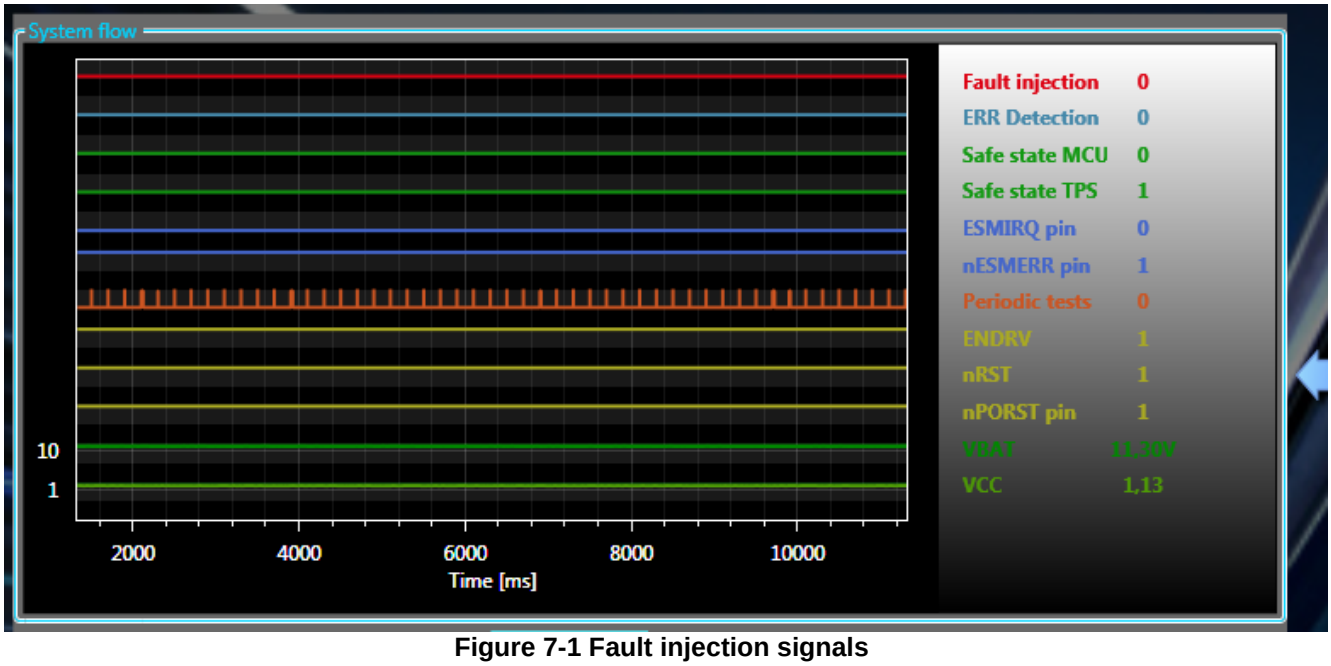

Figure 7-1 Fault injection signals

- **Fault injection**
  GPIO signal is raised by application previous to a fault generation. In normal operation this signal is low indicating that no fault injection is active.
  Meaning: Appearance of a fault

- **ERR Detection**
  GPIO signal set when an error is detected by the safety application firmware. An Error is detected when
  1. An abort handler is called
  2. The ENDRV is detected low
  3. An ESM interrupt is detected
  Meaning: The fault has been detected

- **Safe state MCU**
  A signal that indicates that the MCU is in “Safe state”. The safety MCU is in “Safe state” according to user manual when the PORST input is active low.
  Meaning: Safe State reached

- **Safe state TPS**
  A signal that indicates that the TPS moved the system to safe state. In a safety system all the devices and components connected to ENDRV pin shall enter “safe state”.
  Meaning: Safe State reached

- **ESMIRQ pin**
  GPIO signal set in ESM interrupt handler
  Meaning: The fault has been detected

- **nESMERR pin**
  Signal issued by the ESM of the SDUT
  Meaning: Indication of a fault

- **Periodic tests**
  A GPIO Signal triggered by the application every time when a periodic tests cycle starts. This signal always indicates when the task instance servicing the runtime self-tests starts with a cycle. Which one of the runtime tests are really operated are determined by the runtime test configuration loaded previously through the GUI. So even if no self-test is enabled the signal is raised for a short time anyway.
  The frequency of the signal depends on the safety loop counter setting.
  Meaning: IO signal indicating start of run time tests cycle

- **ENDRV**
  Output signal of the TPS to indicate that the error counter has expired the limit.
  Meaning: Safe State reached

- **nRST**
  Signal of the SDUT to indicate a reset state
  Meaning: System restarts

- **nPORST pin**
  nRES output of the TPS
  Meaning: Power down

- **VBAT**
  power supply of the TPS

- VCC
  Core power supply to the safety MCU device (output from TPS).

A signal name, with a prefixed “n”, means that it is an active low signal. The default value of these signals is high.
The system reaction to specific faults very often is similar. So it makes sense to categorize the typical fault reactions.

7.2.2 *Times measured*

- Fault detection time:
  This is the duration from the start of a “fault injection” until the error is detected. Detection of an error is indicated through the ERR Detection signal. In some cases a reset occurs before the ERR Detection is raised. In these cases the fault detection end timestamp is sampled when the nPORST or nRST gets active.
- Time until safe state entry:
  The duration from the start of a fault injection till the safe state pin is asserted.

7.2.3 *Error servicing mechanism*

If an error is detected by the safety application program and the recovery from the error is not possible the error handler routine “HSA_EnterSafeState(DoReset)” is called. In this handler the “ERR Detection“-signal is asserted. This routine provides two methods to behave controlled through a parameter. One method is that the routine generates a system reset and the firmware restarts.
The other way to handle the error handling is to reside in an endless loop. The safety application while waiting in the interrupt handler does not serve the external watchdog (Q&A). As a consequence the error-counter of the TPS increments above the configured limit leading to a signal indication of the nENDRV pin. At least a power on reset is generated by the external watchdog.

Reasons to call the function “HSA_EnterSafeState ()” are:

- Abort exception
- ESM interrupt
- ENDRV is detected low
- In conjunction with fault injection when a self test failed

7.2.4 *Faults affecting the ESM (group 1 and group 2)*

Since the ESM group1 and group2 errors are handled identical they are assigned to one category.

Category 1 behavior

Many faults force the ESM to generate an interrupt and set the nERROR pin if configured respectively. In the interrupt handler of the safety application the ESMIRQ pin is set. This signal reflects the recognition of the failure. The behavior is identical for group 1 and group 2 errors.
The processing of the ESM interrupt handler depends on the parameter for the respective ESM group recovery setting (refer to 0). If this setting is set to “no” then the “HSA_EnterSafeState ()” function is called. If the parameter is set to “yes” then the system emulates a repair and continues to process.

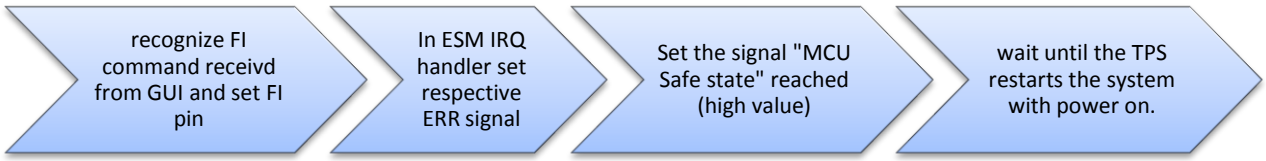
recognize FI command receivd from GUI and set FI pin
In ESM IRQ handler set respective ERR signal
Set the signal "MCU Safe state" reached (high value)
wait until the TPS restarts the system with power on.

Figure 7-2 Data flow of the SDUT no recovery from fault

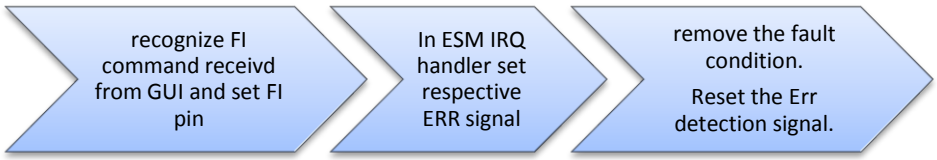
recognize FI command receivd from GUI and set FI pin
In ESM IRQ handler set respective ERR signal
remove the fault condition.
Reset the Err detection signal.

Figure 7-3 Data flow of the SDUT recovery from fault

Note: The safety application while waiting in the interrupt handler does not serve the external watchdog (Q&A). As a consequence the error-counter of the TPS increments above the configured limit leading to a signal indication of the nENDRV pin. There is another mechanism which influences the nENDRV behavior. If the TPS is configured to monitor the nESMERR pin then the nENDRV pin is also driven low after a configured timeout expires. (Global GUI parameter = “ESM low signaling duration”).

The screenshot example is generated with a “PSCON-Lockstep PSCON” fault which is a group 1 error. The configuration parameter for recovery is set to “no”. Also the parameters for “ESM IRQ” and Error pin” are enabled.

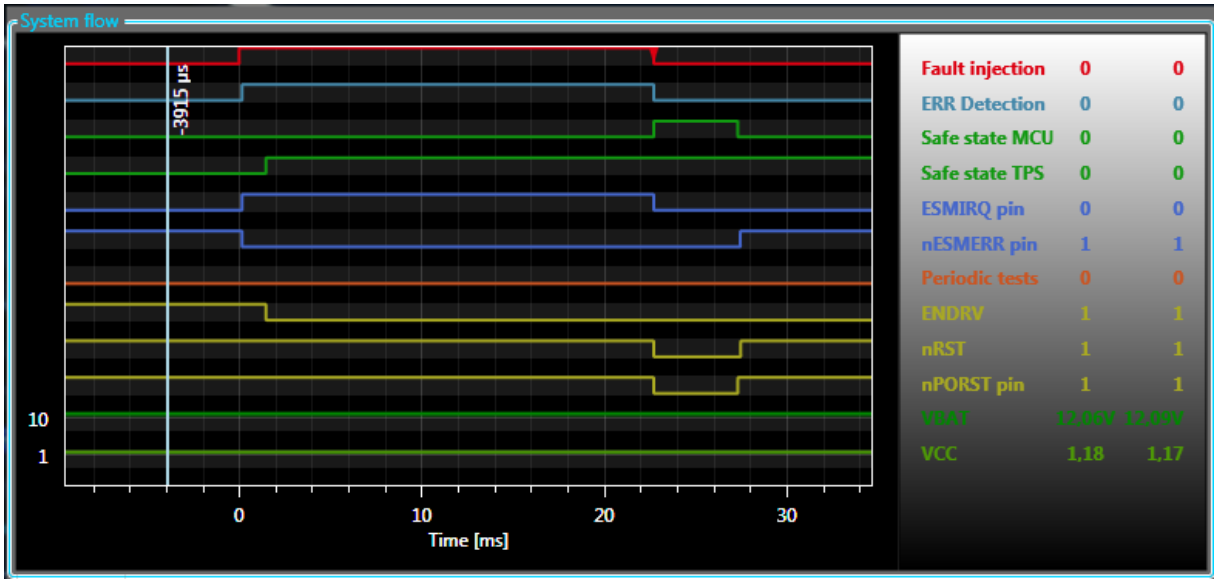

Figure 7-4 Category 1 fault; Lockstep PSCON

The distinctive signals for this flow is that the “ESMIRQ pin” signal is asserted together with the “ERR Detection”-signal.

7.2.5 *Faults which lead to an abort*

Category 2 behavior

These are faults which produce a data abort in the safety application software. In the abort interrupt handler of the safety application the ERRIRQ pin is set. This signal reflects the recognition of the failure. The “HSA_EnterSafeState ()” function is called for every case.

recognize FI command receivd from GUI and set FI pin
In the abort IRQ handler set respective ERR signal
wait until the TPS restarts the systrem with power on.

Figure 7-5 Data flow of the SDUT

Note: Consider that the safety application waiting in the interrupt handler does no more serve the external watchdog (Q&A). As a consequence the error-counter of the TPS increments above the configured limit leading to a signal indication of the nENDRV pin.

The screenshot example shows a “PSON – privileged mode access and program sequence control registers” fault injection.

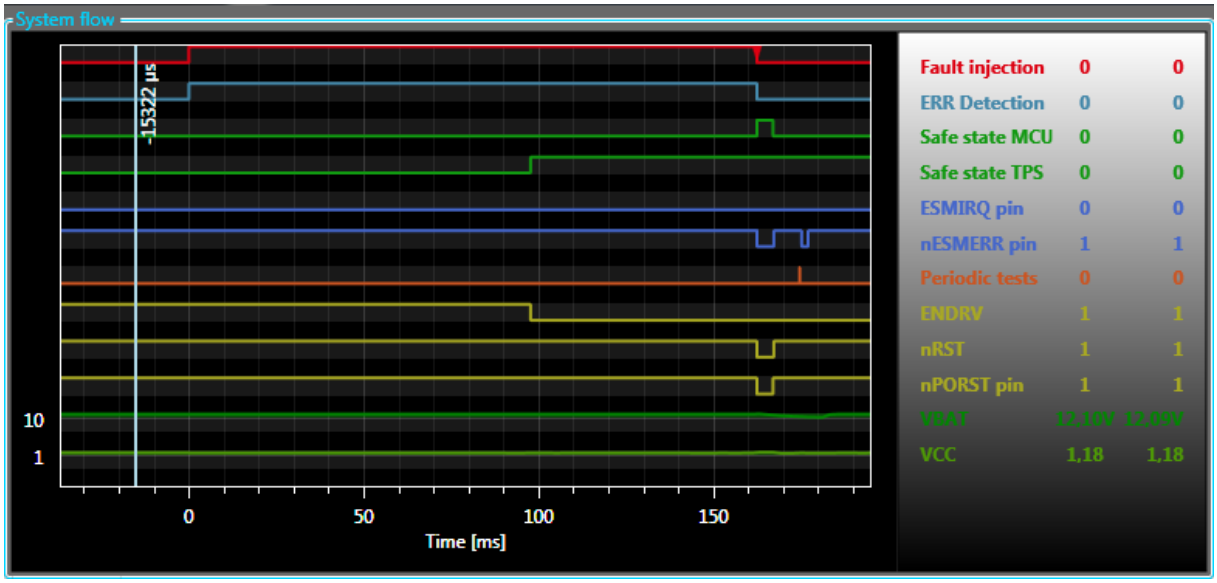

Figure 7-6 Category 2 fault; PMA on PSCON

The distinctive signals for this flow are that the “ESMIRQ pin” signal is not asserted. The “ERR Detection”-signal is asserted just a short time after the fault injection.

7.2.6 *Faults injected on Power rails to TPS*

Category 3 behavior

These are faults affecting the TPS companion chip. Normally with these faults the SDUT gets a power on reset asserted by the TPS device and enters SafeState. The TPS also triggers the ENDRV to indicate to all system components to move to SafeState. After the fault is deserted PORST and nRST are released and the SDUT restarts processing with the boot time tests. The processing of the boot time tests are sensed through the toggling of the nESMERR signal.
The screenshot example shows a “POWER_SUPPLY_SIGNALS-Under voltage on VBAT for TPS.

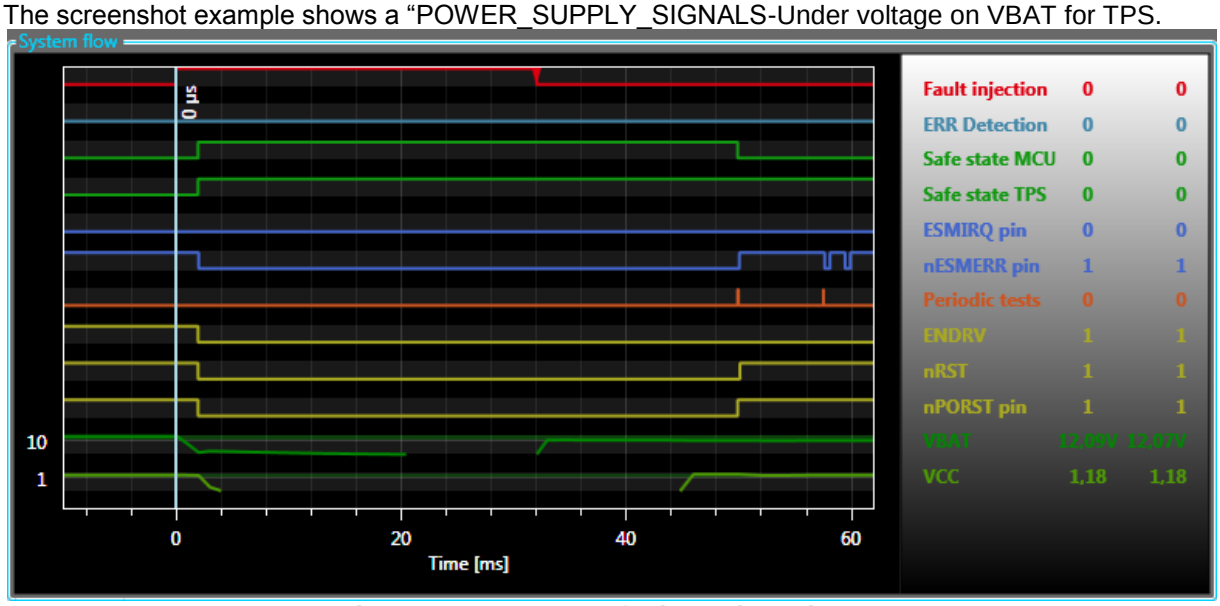

Figure 7-7 Category 3 fault; Under voltage on VBAT

The signal sequence is characterized with an nPORST-signal inverted to the “Fault injection”- signal.

7.2.7 *Faults injected on Power rails and reset lines to SDUT*

Category 4 behavior

These are faults affecting the power supply rails provided by the TPS to the safety device. The MCU device gets into “reset state” indicated by the “nRST”-signal and sets the ESM error pin. The fault is also detected by the TPS monitoring the error pin.

The screenshot example shows a “POWER_SUPPLY_SIGNALS-Disturb Core power supply to safety device (1.2V). On the signal flow you see that the VCC power is cut down.

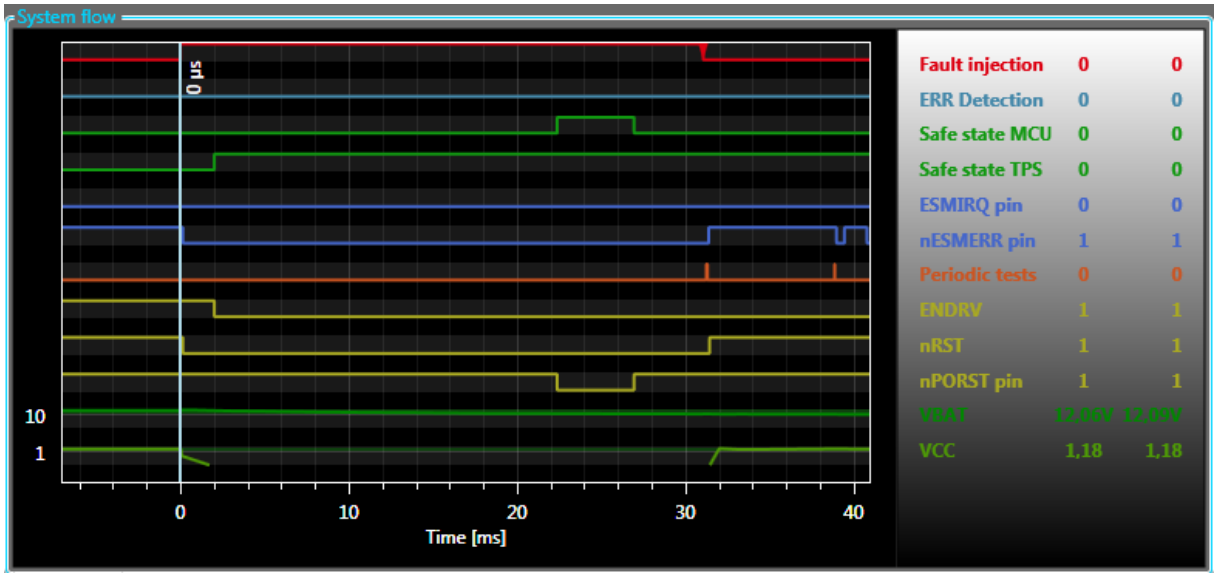

Figure 7-8 Category 4 fault; Disturb CoreVCC 1.2V

The distinctive signal flow is the “nRST”-signal going low together with the “nESMERR pin”-signal and the “ENDRV”- signal.

7.2.8 *Faults affecting the TPS Q&A protocol*

Category 5 behavior

Category 5 faults are the ones which have influence on the TPS question and answer protocol. These are faults which disturb the SPI communication or miss the protocol rules (Q&A).
These faults are detected through the TPS when its internal error counter is increased above 7. Therefore the error detection time is above 100 ms because the wrong data is sent in the “open/close” window cycles.

The screenshot example shows a “EXT_WATCHDOG-Watchdog timer (MCU sends data outside allowed window).

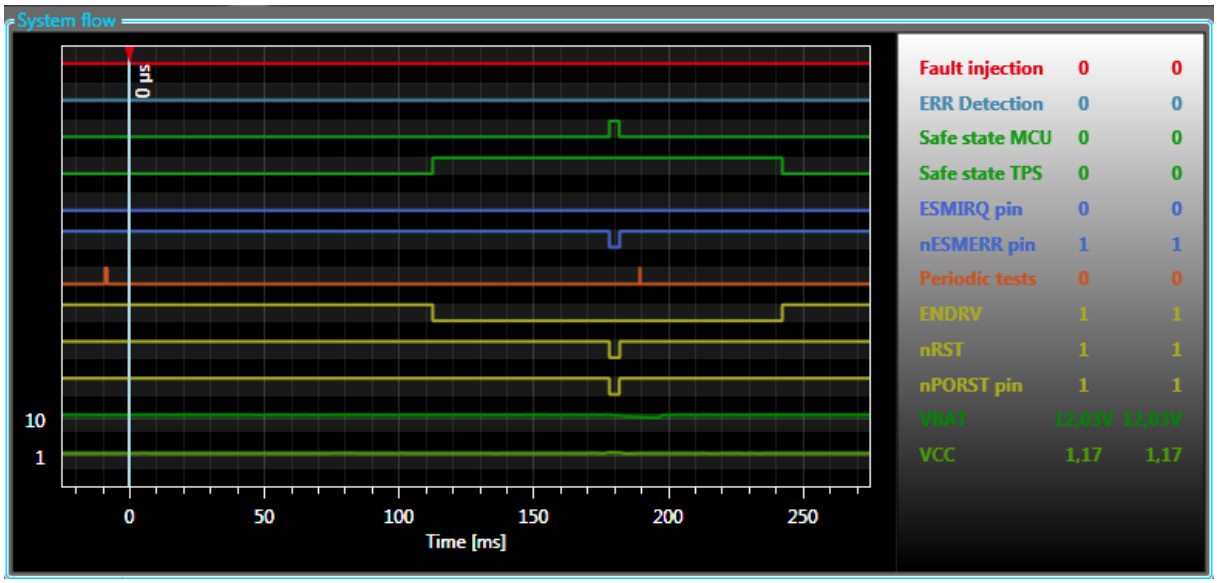

Figure 7-9 Category 5 fault; MCU sends data outside allowed window

The typical signal sequence is characterized with the ENDRV pin driven low which switches the system into safe state. The “Safe state TPS”-signal is raised previous to the signal “Safe state MCU”.

7.2.9 *Faults detected by software application*

Category 6 behavior

These faults are detected through the application software which proceeds the configured “safety tests” at run time or at boot time. These faults neither generate an IRQ on the ESM nor produce an abort. A fault example for a run time test is the periodic CRC check.
For errors detected at run time the “fault detection time” depends on the number of the runtime tests initiated, and is defined as the interval between the fault injection point and the point where the runtime tests detects the fault, provided that the detection occurs within the execution time of the runtime test. So the value of the error detection time depends on the safety loop cycle and the enabled tests.

Some of the faults are injected into the boot time tests. The handling is to store the fault which shall be injected in a variable. After that a “system reset” is generated by software to restart with the boot time tests. The startup code recognizes the request to inject a fault. The “fault detection time” and “Time until safe state entry” of these errors are invalid since the C&M measures the time from the fault injection point to the nRST triggered. For this fault category the measurement method is not capable to retrieve the correct “fault detection time” and “Time until safe state entry”.

The screenshot example shows a “FLASH-Periodic hardware CRC check for Flash contents”.

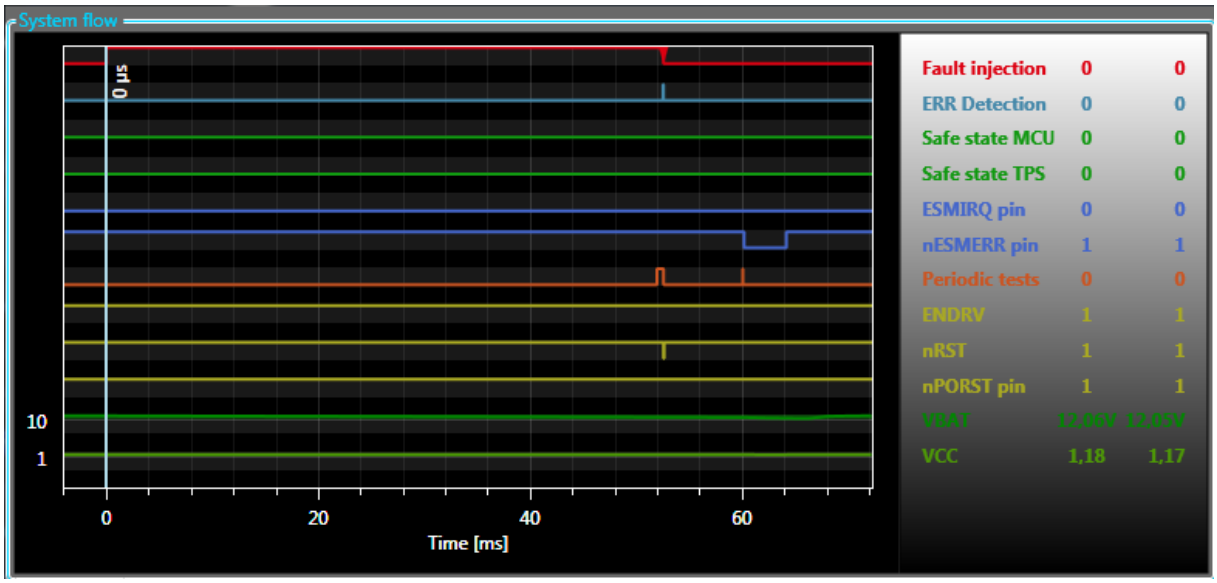

Figure 7-10 category 6 fault; CRC check

The distinctive flow is that the “ERR Detection”-signal is asserted after the assertion of the “Periodic tests”-signal.

A screenshot example of a “FLASH-Boot time hardware CRC check for Flash contents” is shown below.

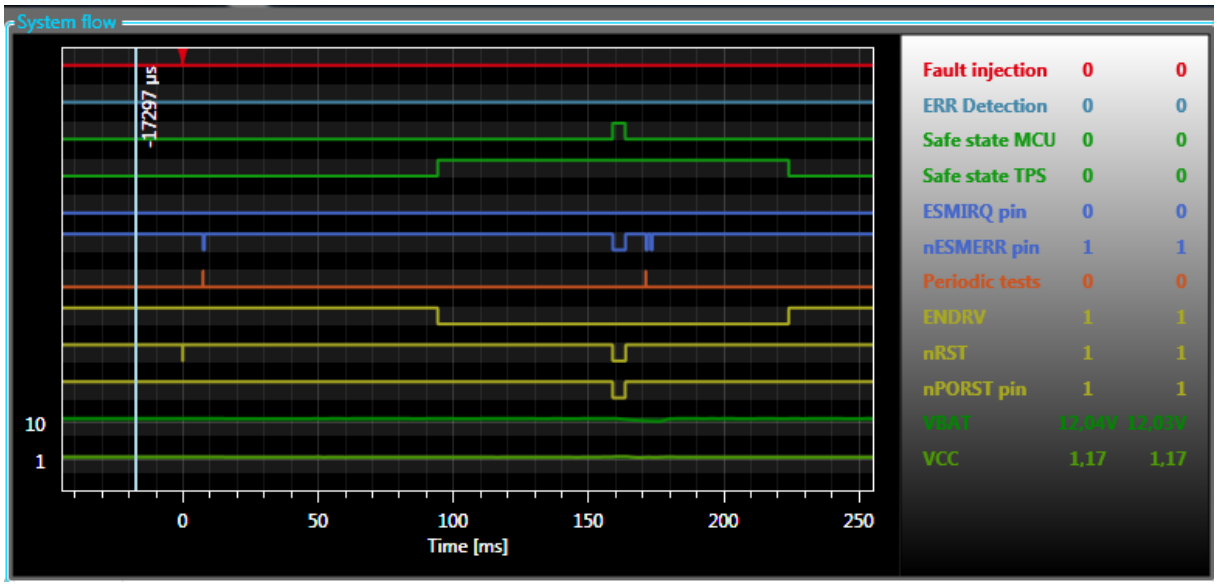

Figure 7-11 category 7 fault; CRC check at boot time

Typical is the “nRST”-signal asserted together with the “Fault Injection”-signal. The error detection occurs during startup. Since the Q&A cannot be served considering the correct timing constraints after the reset the fail counter of the TPS increases and therefore resets the system with POSRT.

7.3 Faults

Explanation of the list entries

Column Fault group:
The Fault group selected in the GUI
Column Fault: Name:

The selected fault to inject
Column Fault: Diagnostic:
This is diagnostic feature referred to the safety manual
Column Fault: Description:
The explanation how a fault is produced
Column Fault: Category:
A Fault category which lead to a typical system behavior
Column Fault: Reaction:
The system behavior (application shall not recover from fault)
Column GUI Parameter:
The GUI Parameters which have influence on the test

Fault group	Fault	GUI Parameter
PSCON	**Name:** Lock step PSCON **Diagnostic:** Lock step PSCON **Description:** The SAFW enables PSCON error forcing mode. **Category:** **1** **Reaction:** An ESM group 1 channel 38 interrupt is generated. In the interrupt handler the "EnterSafeState" function is called if the parameter 'Recover from ESM group 1 error' is set to no. This function waits in an endless loop until the TPS resets the system (ENDRV low, PORST triggered).	• Recover from ESM group 1.
PSCON	**Name:** Privileged mode access and program sequence control registers **Diagnostic:** Privileged mode access and program sequence control registers **Description:** An access mode violation is stimulated. **Category :** **2** **Reaction:** An abort happens and the data abort handler is executed. In the handler the "EnterSafeState" function is called. This function waits in an endless loop until the TPS resets the system (ENDRV low, PORST triggered).	• None
Power supply signals	**Name:** Under voltage on VBAT for TPS **Diagnostic:** The safety device system is controlled by an external device (TPS). **Description:** The power supply VBAT for the TPS is reduced below 4.8Volt **Category:** **3** **Reaction:** The nRST and PORST are triggered to the MCU. The TPS activates "Safe state" indicated with ENDRV signal driven low.	• None

Fault group	Fault	GUI Parameter
Power supply signals	**Name:** Disturb VBAT on TPS **Diagnostic:** The safety device system is controlled by an external device (TPS). **Description:** The power supply VBAT for the TPS is cut off. **Category:** **3** **Reaction:** The nRST and PORST are triggered to the MCU. The TPS activates "Safe state" indicated with ENDRV signal driven low.	• None
Power supply signals	**Name:** Disturb VBAT Safing to TPS **Diagnostic:** The safety device system is controlled by an external device (TPS). **Description:** The VBAT Safing reference to the TPS is cut. **Category:** **3** **Reaction:** The PORST is triggered through the TPS watchdog. The TPS activates “Safe state” indicated with ENDRV signal driven low.	• None
Power supply signals	**Name:** Disturb power supply to safety device (3.3V) **Diagnostic:** external voltage supervisor **Description:** The 3.3V power supply for the safety device is switched down. **Category:** **4** **Reaction:** The nRST is triggered which causes the TPS to lose synchronization (ENDRV low). The TPS triggers a PORST to the safety MCU.	• None
Power supply signals	**Name:** Disturb core power supply to safety device (1.2V) **Diagnostic:** nRST monitoring with TPS **Description:** The 1.2V power supply for the safety device is switched down. **Category:** **4** **Reaction:** Since this is the core power supply the nRST and nERROR is triggered followed by a PORST trigger through the TPS.	• None
External WD	**Name:** Disturb TPS communication: SPI SOMI **Diagnostic:** External watchdog supervision **Description:** The SOMI signal to TPS is disturbed **Category:** **4** **Reaction:** Since the question and answer protocol fails caused by the disruption the internal error counter of the TPS is increased above the configured threshold level. Then the TPS enters safe state (ENDRV low) and restarts the system with power on reset.	• None

Fault group	Fault	GUI Parameter
External WD	**Name:** Stuck on ENDRV PIN (low) **Diagnostic:** External watchdog pin (ENDRV) monitoring **Description:** The ENDRV Pin is forced to low level **Category:** **5** **Detection:** Through polling of the ENDRV pin by the safety application the error is detected. **Reaction:** The "EnterSafeState" function is called. This function waits in an endless loop until the TPS resets the system (ENDRV low, PORST triggered).	• None
External WD	**Name:** Watchdog error failures (MCU sends wrong data) **Diagnostic:** External watchdog **Description:** The safety device sends several wrong answers to the request (Q&A). The timing constraints are considered. **Category:** **5** **Reaction:** The TPS enters reset state after the fail counter of the TPS is increased above the threshold level. Then the TPS restarts the system with power on reset.	• None
External WD	**Name:** Watchdog error failures (MCU sends data outside allowed window) **Diagnostic:** External watchdog supervision **Description:** The safety device sends an answer to the request (Q&A) outside the open window. **Category:** **5** **Reaction:** Since the question and answer protocol fails caused by the timing violation the internal error counter of the TPS is increased above the configured threshold level. Then the TPS restarts the system with power on reset.	• Open window time
Clock	**Name:** Low power clock detection **Diagnostic:** Low power clock detection **Description:** The clock source for the safety device is provided by the C&M device via the “eclock” output signal. This clock source is cut off. **Note:** The parameter “Recover from ESM group error” is not effective for this fault since the fault is generated by the external C&M device. **Category:** **1** **Detection:** The on chip clock monitor detects this error. **Reaction:** An ESM group 1 interrupt is generated. In the interrupt handler the "EnterSafeState" function is called. This function waits in an endless loop until the TPS resets the system (ENDRV low, PORST triggered).	• ESM channel configuration

Fault group	Fault	GUI Parameter
Clock	**Name:** Dual clock comparator **Diagnostic:** Dual clock comparator **Description:** The clock source for the safety device is provided by the C&M device via “eclock” output signal. The frequency of this clock is reduced from 16 MHz to 13.3 Mhz. **Note:** The parameter “Recover from ESM group error” is not effective for this fault since the fault is generated by the external C&M device. **Category:** **1** **Detection:** The DCC module detects this failure at least after 5 ms and activates the corresponding ESM channel. **Reaction:** An ESM group 1 interrupt is generated. In the interrupt handler the "EnterSafeState" function is called. This function waits in an endless loop until the TPS resets the system (ENDRV low, PORST triggered).	• ESM channel configuration
RESET	**Name:** Stuck on nPORST low **Diagnostic:** **Description:** The power on signal is stuck at low. **Category:** **4** **Reaction:** The SDUT remains in reset state since the nPORST is held low.	•
RESET	**Name:** Watchdog error failures (nRST PIN stuck at low) **Diagnostic:** **Description:** The nRST signal is stuck at low. **Category:** **4** **Reaction:** The safety device under test is held in system reset.	•
MCU_SYSTEM	**Name:** Privileged mode access and multi bit key enable **Diagnostic:** Privileged mode access and multi bit key enable **Description:** An access violation is forced by the application software. **Category:** **2** **Reaction:** The data abort handler is entered. The "EnterSafeState" function is called. The function waits in an endless loop until the TPS resets the system (ENDRV low, PORST triggered).	•

Fault group	Fault	GUI Parameter
ESM	**Name:** External MCU Error Pin Monitor (SDUT recovers in time) **Diagnostic:** Error pin monitoring **Description:** The Error Pin is raised for a short time (200 us). This shall simulate that the safety application has detected an error and resolves it in time. **Category:** **1** **Reaction:** The ESMError pin is active for 200 microseconds. The Error is detected and resolved.	• Monitor safety device under test
ESM	**Name:** External MCU Error Pin Monitor (SDUT does not recover in time) **Diagnostic:** Error pin monitoring **Description:** The error pin is raised for a specific time longer than that of the external watchdog timeout for monitoring the MCU error pin. **Category:** **1** **Reaction:** The TPS detects the error through the MCU error pin monitoring feature. The TPS resets the system (ENDRV low, PORST triggered)..	• Monitor safety device under test (TPS) • Error pin low signaling duration (TPS)
ESM	**Name:** Software test of error path reporting **Diagnostic:** Software test of error path reporting **Description:** A TCM RAM self test function (error forcing 1 Bit) is called. Due to that function call the activation of the MCU Error pin is expected. The MCU Error pin is checked and a fault detection emulated by the safety application. **Category:** **1** **Reaction:** The application software detects the failure on the error pin and calls the "EnterSafeState" function. The function waits in an endless loop until the TPS resets the system (ENDRV low, PORST triggered).	•
CPU	**Name:** Lock step compare **Diagnostic:** Lock step compare **Description:** Enable CCM-R4 error forcing mode **Category:** **1** **Reaction:** An ESM group 1 interrupt is generated. In the interrupt handler the "EnterSafeState" function is called if the parameter 'Recover from ESM group 1 error' is set to no. The function waits in an endless loop until the TPS resets the system (ENDRV low, PORST triggered).	• ESM channel configuration • Recover from ESM group 1

Fault group	Fault	GUI Parameter
CPU	**Name:** Periodic execution of STC **Diagnostic:** Periodic execution of STC **Description:** The STC test is called with a time window given which is too short. **Category:** **6** **Note:** The parameter “Recover from ESM group error” is not effective for this fault since the STC run time test occurs a reset. **Reaction:** The STC self-test is called and returns with an error. The error is recognized by the application and then the "EnterSafeState" function is called. The function waits in an endless loop until the TPS resets the system (ENDRV low, PORST triggered).	• ESM channel configuration
CPU	**Name:** Illegal operation and instruction trapping **Diagnostic:** Illegal operation and instruction trapping **Description:** Force an access violation **Category:** **2** **Reaction:** An undefined instruction exception happens and the exception handler is executed. In the handler the "EnterSafeState" function is called. This function waits in an endless loop until the TPS resets the system (ENDRV low, PORST triggered).	• None
FLASH	**Name:** Flash data ECC **Diagnostic:** Flash data ECC **Description:** An ECC error is forced **Category:** **2** **Reaction:** The access to data with wrong ECC bits lead to a data abort and the data abort handler is called. In the handler the "EnterSafeState" function is called. This function waits in an endless loop until the TPS resets the system (ENDRV low, PORST triggered).	• Flash Data ECC logic
FLASH	**Name:** Address parity **Diagnostic:** ATCM Address Bus Parity. The on-chip ATCM bus connection to Flash memory is supported by a parity diagnostic on the address signals. **Description:** Force an address parity fault by calling the safety™ diagnostics Library API function “Sl_SelfTest_Flash” with the parameter “FLASH_ADDRESS_PARITY_FAULT_INJECT”. **Category** **1** **Reaction:** An ESM group 1 interrupt is generated. In the interrupt handler the "EnterSafeState" function is called if the parameter 'Recover from ESM group 1 error' is set to no. This function waits in an endless loop until the TPS resets the system (ENDRV low, PORST triggered).	• ESM channel configuration • Recover from ESM group 1

Fault group	Fault	GUI Parameter
FLASH	**Name:** Boot time hardware CRC check for Flash contents. **Diagnostic:** CRC check **Description:** The CRC value to compare against is modified and then a reset is generated. The error is detected at boot time. **Category:** **6** **Reaction:** The boot time CRC check detects the error and calls the function "EnterSafeState". This function waits in an endless loop until the TPS resets the system. The ERR Detection signal is not sensitive to this error, since the fault detection occurs before the signal's GPIO pin is initialized.	• None
FLASH	**Name:** Periodic hardware CRC check for Flash contents. **Diagnostic:** CRC check **Description:** The CRC value to check against is modified. Consider that the runtime checking for CRC test has to be activated. **Category:** **6** **Reaction:** The periodic CRC run time test detects the error. The safety application firmware calls the function "EnterSafeState". This function initiates a system reset, which causes the TPS to lose synchronization (ENDRV low). The TPS triggers a PORST to the safety MCU. The fault detection time depends on the number of the runtime tests initiated, and is defined as the interval between the fault injection point and the point where the runtime tests detects the fault, provided that the detection occurs within the execution time of the runtime test.	• CRC calculation
FEE	**Name:** Data ECC **Diagnostic:** FEE Data ECC **Description:** An ECC error is forced **Category:** **1** **Reaction:** An ESM group 1 interrupt is generated. In the interrupt handler the "EnterSafeState" function is called if the parameter 'Recover from ESM group 1 error' is set to no. This function waits in an endless loop until the TPS resets the system (ENDRV low, PORST triggered).	• ESM channel configuration • Recover from ESM group 1
SRAM	**Name:** Data ECC **Diagnostic:** SRAM Data ECC **Description:** An ECC error is forced on the SRAM **Category:** **2** **Reaction:** The access to data with wrong ECC bits lead to a data abort and the data abort handler is called. In the handler the "EnterSafeState" function is called. This function waits in an endless loop until the TPS resets the system (ENDRV low, PORST triggered).	• RAM Data ECC logic

Fault group	Fault	GUI Parameter
SRAM	**Name:** Correctable ECC profiling **Diagnostic:** Correctable ECC profiling **Description:** The self-test library function is called which forces the fault. **Category:** **1** **Reaction:** An ESM group 1 interrupt is generated. In the interrupt handler the "EnterSafeState" function is called if the parameter 'Recover from ESM group 1 error' is set to no. This function waits in an endless loop until the TPS resets the system (ENDRV low, PORST triggered)..	• ESM channel configuration • Recover from ESM group 1
SRAM	**Name:** Redundant address decode **Diagnostic:** Redundant address decode **Description:** The self-test library function is called which forces the fault. **Category:** **1** **Reaction:** An ESM group 2 interrupt is generated. In the interrupt handler the "EnterSafeState" function is called if the parameter 'Recover from ESM group 1 error' is set to no. This function waits in an endless loop until the TPS resets the system (ENDRV low, PORST triggered).	• ESM channel configuration • Recover from ESM group 2
SRAM	**Name:** Boot time PBIST check of RAM **Diagnostic:** PBIST check **Description:** A fault is stored which indicates to the startup code to inject a fault after the reset is processed. The boot time self-test then injects the fault starting a PBIST-check (programmable built in self-test) with a wrong algorithm parameter. **Category:** **6** **Reaction:** The safety application firmware checks the test result which is fail and calls the function "EnterSafeState". This function waits in an endless loop until the TPS resets the system. The ERR Detection signal is not sensitive to this error, since the fault detection occurs before the signal's GPIO pin is initialized.	•
SRAM	**Name:** Periodic PBIST check of RAM **Diagnostic:** PBIST check **Description:** The pBIST is called with a wrong algorithm parameter. **Category:** **6** **Reaction:** The safety application firmware calls the function "EnterSafeState". This function initiates a system reset, which causes the TPS to lose synchronization (ENDRV low). The TPS triggers a PORST to the safety MCU.	•

Fault group	Fault	GUI Parameter
VIM	**Name:** VIM SRAM data parity **Diagnostic:** VIM SRAM data parity **Description:** The VIM RAM parity feature is enabled and a backup interrupt handler is setup. Then one parity bit of an entry is flipped. After that an access to that entry is done. **Category:** **1** **Reaction:** The MCU error pin gets active (low). Additionally the backup interrupt handler is called. The implementation of the backup handler calls the "EnterSafeState" function. This function waits in an endless loop until the TPS resets the system (ENDRV low, PORST triggered).	• ESM channel configuration • Recover from ESM group 1
ADC	**Name:** Boot time input self-test **Diagnostic:** ADC input self-test **Description:** A fault is stored which indicates to the startup code to inject a fault after the reset is processed. The boot time self-test then injects the fault by selecting a free ADC channel. This will produce an error similar to a stuck-at fault at an input channel. **Category:** **6** **Reaction:** The boot time ADC input self-test detects the error. The safety application firmware calls the function "EnterSafeState". This function waits in an endless loop until the TPS resets the system. The ERR Detection signal is not sensitive to this error, since the fault detection occurs before the signal's GPIO pin is initialized.	•
ADC	**Name:** Periodic time input self-test **Diagnostic:** ADC input self-test **Description:** A fault is stored which indicates to the periodic tests to inject a fault. The periodic ADC self-test then injects the fault by selecting a free ADC channel. This will produce an error similar to a stuck-at fault at an input channel. Ensure that the Safety Diagnostics parameter for periodic ADC self-test is enabled. It is recommended to set the Safety loop to 100 ms. **Category:** **6** **Reaction:** The periodic ADC input self-test detects the error. The safety application firmware calls the function "EnterSafeState". This function initiates a system reset, which causes the TPS to lose synchronization (ENDRV low). The TPS triggers a PORST to the safety MCU. The fault detection time depends on the number of the runtime tests initiated, and is defined as the interval between the fault injection point and the point where the runtime tests detects the fault, provided that the detection occurs within the execution time of the runtime test.	•

Figure 7-12 Faults

8 Profiling

The profiling feature provides a mechanism to measure the execution time of several safety diagnostics. This could help a customer to design their own software application related to safety runtime tests. There are specific predefined tests along with others which depend on parameter configuration.
Select the “Profiling” tab to open the window “Validating & Profiling” by selecting the “Profiling” tab. Depending on the selected test-group (e.g. PSCON) one specific test can be selected for measuring the execution time. The test is executed once if the Profiling button is clicked.

8.1 Data flow

If a profiling measurement is started using the GUI, a command with a unique ID related to the selected test is sent via the GUI to the C&M device, which then forwards the command to the SDUT. The SDUT recognizes the profiling command and raises a PIN that is detected by the C&M device to start the time stamping of the test. After that, the SDUT calls the corresponding test function by the safety library. Returning from the safety library, the profiling PIN is cleared and the safety device continues operation (refer to chapter 10.5.6.2).
The C&M device samples the signal and generates a timestamp. This information is sent to the GUI, where it is visualized.

8.1.1 *Understanding what is measured*

The sampling of the timestamps related to the profile signal is done within the C&M device. A timestamp is taken when the C&M device recognizes a signal level change issued by the SDUT device. The timestamp together with the sampled signal is sent to the GUI.
Since the profile signal drives an interrupt on the GPIO input of the C&M device, the timestamp value is captured in the interrupt handler. A message is generated containing the signal level and the timestamp sampled in IRQ. The GUI then displays the signal sequence and calculates the measured profiling time.
Since this process is done with software, the resolution is limited. It has to be noted that tests with execution time smaller than 5 µs might not be recognized. These tests will be executed 10 times consecutively, and as such are marked with “(x10)” at their name. For these tests, the measured duration is automatically divided by 10. Take into account that the accuracy is about 1 µs.

recognize command receivd from GUI
forward a profile message to SDUT
IRQ handler for profile signal samples timestamp
Signal message with sampled timestamp is sent to GUI
IRQ handler for profile signal samples timestamp
Signal message with sampled timestamp is sent to GUI

Figure 8-1: Data flow of C&M device

The safety application is in charge of controlling the Profiling Signal and executing the self-test. This is done by signaling the begin of test by pulling the Profiling Signal high, and then calling a specific function from the safety library, which in turn starts the hardware test. The safety library function then waits until the MCU completes the test. It returns the test result to the application, which in turn signals end of test by pulling the Profiling Signal low.

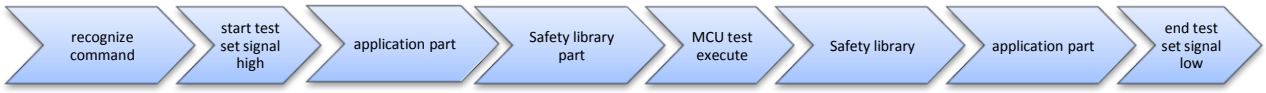

Figure 8-2: Data flow of safety device

8.1.2 *Special Considerations*

Since the test execution is operated in a software context, several issues need to be considered:

- The task which runs the test (HSA_SafetyLibServer Task) is set to highest priority (prior to the test execution), so that it does not get interrupted by other tasks. This may lead to the Q&A task running out of time, if the test takes too much time to execute. However, this has no influence on the measured result.
- Profiling depends highly on the clock frequency, with which the safety MCU is clocked to run the safety application, which in turn runs the self-tests. The PLL is configured to generate a clock of 160MHz which is used as the HCLK source, and VCLK = HCLK / 2.
- Interrupts from peripherals may occur during the test run and may add a small part to the measured time. The only interrupt sources are the MIBSPI1 module and the RTI counter. Important is to keep in mind that the MIBSPI 1 interrupt is driven through the C&M and occurs every 20 ms.
- Some self-tests can be parameterized arbitrarily, and may cause the test to fail if conflicts occur with the set parameters. If the test fail is detected, then the measured profiling time is invalidated and is indicated in the HSK-Monitor GUI as shown below

Figure 8-3: Profile Timing invalid

- It is strongly recommended that the parameter “System load” is set to a value of 0 (slider in global settings page) to avoid an influence through the RTI interrupt.

8.2 Profiling Tests

8.2.1 *Specific details for the tests*

Taking a look into the list of tests, you can see that each test group has predefined tests, and at least one test depends on GUI parameter configuration.

8.2.1.1 Dedicated tests calling the self-test in the safety library directly

- **Category 1 tests**

These tests directly call the corresponding API function provided with the safety library with a specific parameter set. This parameter set is defined through the test itself. The application itself does nothing more.
Example:
Group PSCON, Test: error forcing test: Stuck on error signal

8.2.1.2 Run time parameterized tests

- **Category 3 tests**

These tests consider the GUI parameter settings (parameter settings are explained in the chapter HSK Monitor). The test routine considers which tests are enabled and then executes them one after each other. These profiled tests depend on the GUI parameter configuration which provides enabling/disabling the tests. Profiling such a test means that the runtime test routine for the selected test group is called.
Example:
Group Flash, Test: Parameterized Flash self-test

- **Category 4 tests**

This test differs from the other ones because the test itself does a reset. Therefore the application saves the core register set and other registers previously to execute the test. When the reset occurs the saved registers are restored to be able to continue with the application. It should be considered that the time for saving and restoring the registers is included in the result time.
Since this tests is also dependent from the interval counter we recommend to profile several measurements with different interval counter values.

Example:
Group STC, Test: Parameterized STC Test (Logic Built in Self Test)

Figure 8-4: Data flow of safety device

- **Category 5 tests**

This test is executed on exactly one RAM space selected with the parameter “PBIST GROUP”. The test algorithm is selected with the two parameters “algorithm” and “memory type”.
Since the test destroys the RAM contents it could configured if the RAM content shall be saved previous to the test execution and restored after the test execution. This is done with the “Store/Restore selected RAM” parameter. The time for saving and restoring is included in the profile time.
It is the responsibility of the user to ensure that a valid configuration is selected. If the test execution detects an error the profile time is marked invalid.
Example:
Group pBIST, Test: Parameterized programmable built in self-test

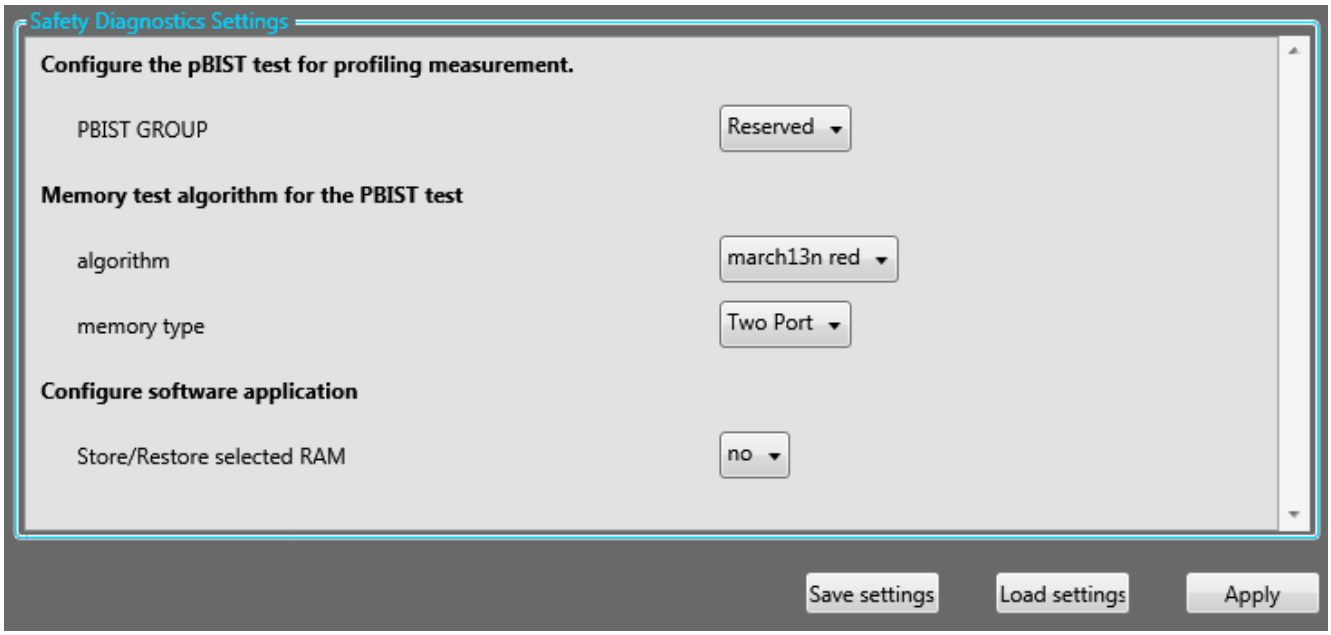

Figure 8-5: Parameter set for the pBIST test

8.2.2 *Profiling Tests list*

The following is a list of tests that are provided by the GUI:

Explanation of the list entries

Column Test group:
The Test group selected in the GUI
Column Test: Name:
The selected tests to execute
Column Test: Call:
This is the function call, the execution time of which is to be measured. Function calls into the safety library start with the prefix “SL_”; function calls in the safety application start with “HSA_”.
Column Test: Test type:
A parameter which identifies the test for the safety diagnostics library API call.
Column Test: Category:
The test category mentioned above.
Column Parameter:
The GUI Parameters which have influence on the test.

Test group	Test		GUI Parameter
PSCON	**Name:** **Call:** **Test type:** **Category:**	Error forcing test: Stuck on error signal SL_SelfTest_PSCON(…) PSCON_ERROR_FORCING 1	• none
PSCON	**Name:** **Call:** **Test type:** **Category:**	Error forcing test: Error signal out SL_SelfTest_PSCON(…) PSCON_SELF_TEST_ERROR_FORCING 1	• System load
PSCON	**Name:** **Call:** **Test type:** **Category:**	Error forcing test: Lockstep SL_SelfTest_PSCON(…) PSCON_SELF_TEST 1	• System load
PSCON	**Name:** violation **Call:** **Test type:** **Category:**	Error forcing test: Privilege mode access SL_SelfTest_PSCON(…) PSCON_PMA_TEST 1	• System load

Test group	Test		GUI Parameter
PSCON	**Name:** **Call:** **Test type:** **Category:**	All parameterized tests together HSA_SLS_PSCON_RuntimeTest() several tests may be called depending on the enabled tests. 3	• Run time test configuration • System load
STC	**Name:** **Call:** **Test type:** **Category:**	Parameterized STC test (logic built in self-test) HSA_SLS_STC_Runtime_Test() ST_RUN 4	• Run time test configuration • Interval count • Run timeout • System load
pBIST	**Name:** **Call:** **Category:**	Parameterized programmable built in self-test HSA_SLS_pBistRam(…) 5	• Run time test configuration • Algorithm • Port • Save/Restore • System load
FLASH	**Name:** **Call:** **Test type:** **Category:**	Error forcing test: 1 Bit error SL_SelfTest_Flash (…) FLASH_ECC_TEST_MODE_1BIT 1	• System load
FLASH	**Name:** **Call:** **Test type:** **Category:**	Error forcing test: 2 Bit error SL_SelfTest_Flash (…) FLASH_ECC_TEST_MODE_2BIT 1	• System load
FLASH	**Name:** **Call:** **Category:**	Parameterized Flash self-test HSA_SLS_Flash_RuntimeTest () 3	• Error forcing 1 Bit • Error forcing 2 Bit • System load
FLASH	**Name:** **Call:** **Category:**	CRC calculation on code SL_CRC_Calculate(…) 3	• Start address = code start address • End address = code end address • System load
FLASH	**Name:** **Call:** **Category:**	Parameterized CRC calculation SL_CRC_Calculate(…) 3	• Start address • End address • System load

Test group	Test		GUI Parameter
SRAM	**Name:** **Call:** **Test type:** **Category:**	Error forcing test: 1 Bit error SL_SelfTest_SRAM(…) SRAM_ECC_ERROR_FORCING_1BIT 1	• System load
SRAM	**Name:** **Call:** **Test type:** **Category:**	Error forcing test: 2 Bit error SL_SelfTest_SRAM(…) SRAM_ECC_ERROR_FORCING_2BIT 1	• System load
SRAM	**Name:** **Call:** **Test type:** **Category:**	Error forcing test: Address and control parity SL_SelfTest_SRAM(…) SRAM_PAR_ADDR_CTRL_SELF_TEST 1	• System load
SRAM	**Name:** **Call:** **Test type:** **Category:**	Error forcing test: Redundant address decode SL_SelfTest_SRAM(…) SRAM_RADECODE_DIAGNOSTICS 1	• System load
SRAM	**Name:** **Call:** **Category:**	Parameterized SRAM tests HSA_SLS_SRAM_RuntimeTest() 3	• Run time test configuration • System load
SRAM	**Name:** **Call:** **Category:**	CRC calculation on VIM RAM SL_CRC_Calculate(…) 1	• System load
SRAM	**Name:** **Call:** **Category:**	Parameterized CRC calculation SL_CRC_Calculate(…) 3	• Start address • End address • System load
EFUSE	**Name:** **Call:** **Test type:** **Category:**	Error forcing test: ECC SL_SelfTest_EFUSE(…) EFUSE_SELF_TEST_ECC 1	• System load
EFUSE	**Name:** **Call:** **Test type:** **Category:**	Error forcing test: stuck at zero SL_SelfTest_EFUSE(…) EFUSE_SELF_TEST_STUCK_AT_ZERO 1	• System load

Test group	Test		GUI Parameter
EFUSE	**Name:** **Call:** **Category:**	Parameterized EFUSE tests HSA_SLS_EFUSE_RuntimeTest() 3	• Run time test configuration • System load
CCMR4	**Name:** **Call:** **Test type:** **Category:**	CPU lockstep SL_SelfTest_CCMR4F(…) CCMR4F_SELF_TEST 1	• System load
CCMR4	**Name:** **Call:** **Test type:** **Category:**	CPU Error forcing test SL_SelfTest_CCMR4F(…) CCMR4F_ERROR_FORCING_TEST 1	• System load
CCMR4	**Name:** **Call:** **Test type:** **Category:**	CPU self-test error forcing SL_SelfTest_CCMR4F(…) CCMR4F_SELF_TEST_ERROR_FORCING 1	• System load
CCMR4	**Name:** **Call:** **Category:**	Parameterized CPU-R4 tests HSA_CCMR4_RuntimeTest() 3	• Run time test configuration • System load
ADC	**Name:** **Call:** **Category:**	ADC self-test Conversion SL_SelfTest_ADC(…) 1	• ADC = ADC1

Table 8-1 Profiling tests

8.2.3 *Profiling full safety task*

There is another test, which measures the execution time of all tests being executed during runtime. This test is started with the button “Profiling full safety task”. The execution time depends very much on the enabled and parameterized run time configuration.
The “Profiling full safety task” duration depends highly on the implementation of the firmware application. The SAFW implementation encapsulates the execution of the configured runtime tests in a separate task. In this task, each test is assigned to an execution step. The “Profiling full safety task” includes the execution process of all steps, executed one after the other. Since this implementation includes more software execution time, the measurement duration is longer than the cumulative summary of the single run time tests. Refer to chapter 10.5.6.2 for a closer look into it.

9 Application example demonstration

The kit includes a few applications just for demonstration purposes. The user benefit is a “how to” integrating some application code into the safety software framework (safety application).
The following application examples are:

- Temperature sensor
- Push buttons
- Onboard display
- Accelerometer
- User controlled LEDs

Beside these board components demonstration a task monitor is realized using GPIOs.

Most of the application example code is collected in the file “HSA_SensorAndHMI.c” and it is built as a task instance (further simply called “sensor task”) running periodically with low priority.

9.1 Temperature sensor

The SafeTI™-HSK features a simple 100KOhm NTC thermistor as an ambient temperature sensor. It is connected to the input channel 1 of the ADC 1 peripheral of the safety MCU.
When the sensor task is started the ADC is initialized and the conversion is started. The value is read out and then the conversion is restarted again. The sensor value (voltage) if different from the previous one is then forwarded via the C&M device to the GIU. The GUI calculates the temperature depending on the parameters determined by the NTC and visualizes the result on the “Application page”.

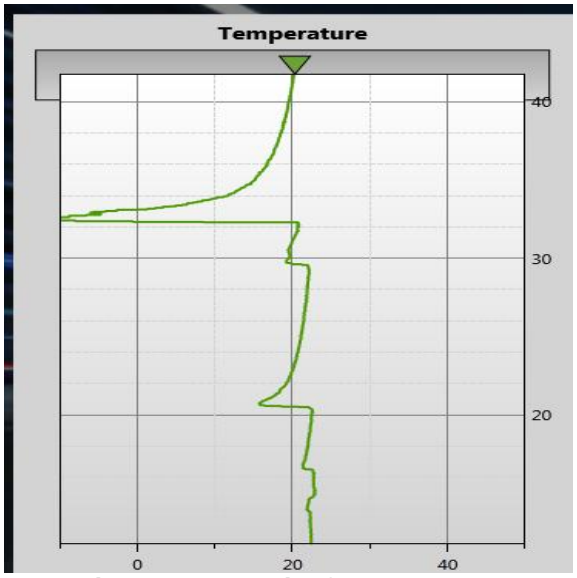

Figure 9-1: March of temperature

Temperature calculation

The formula retrieved from data sheet

$$B = \frac{\ln\left(\frac{R}{R25}\right)}{\left(\frac{1}{t} - \frac{1}{T25}\right)}$$

In our case we can use voltage instead of resistance

$$B = \frac{\ln\left(\frac{V25}{V}\right)}{\left(\frac{1}{t} - \frac{1}{T25}\right)}$$

$V25 = 3.3V/2 = 1.65V$
$T25 = 273+25 = 298\ °K$

The formula for the current temperature calculation is

$$T = \frac{1}{\frac{1}{298} + \frac{\ln(1.65)}{4250}}$$

9.2 Accelerometer

The SafeTI™-HSK also features a small, thin, ultralow power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ±16 g. The digital output data is formatted as 16-bit twos complement and is accessible to the safety MCU through a 4-wire SPI digital interface. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (3.9 mg/LSB) enables measurement of inclination changes less than 1.0°.

The safety application reads out the accelerometer data periodically also in the sensor task. It is forwarded via the C&M device to the GUI. The data includes a timestamp which is set when the data is read out via MIBSPI from the accelerometer device.
The GUI visualizes the result on the “Application page”. There are 3 acceleration axes displayed x (horizontal), y (horizontal orthographic to x) and z vertical. The values are standardized to acceleration due to gravity.

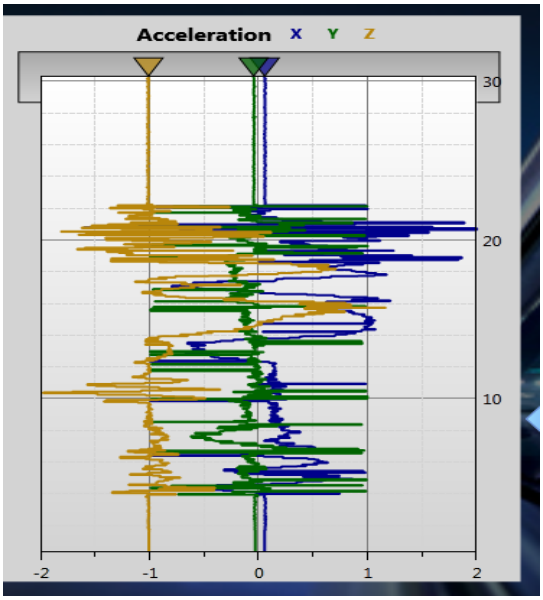

Figure 9-2: Acceleration move

9.3 Onboard display

The SafeTI™-HSK features an on board 128 x 32 pixels LCD with white LED backlight. The display is connected to the safety MCU via SPI2 port.

The software implementation provides an interface to all instances which may output one of two predefined pictures or simply text. One picture is the Hitex logo and the other one is the logo of Texas Instruments.

The display output is realized as a task instance with an input queue containing the text messages. This prevents the interruption of a text output initiated by a low priority task through a task with higher priority.

To test the picture output press the user buttons. A text can be activated using the GUI on the user command page executing a user command.

9.4 Push buttons

Four push buttons are provided. The buttons are polled in the HSA_RunStateCb() function. This is a call-back function which is processed each time the safeRTOS calls the scheduler. The state of the buttons is saved and later on investigated in the sensor task.

User button 1 (SW500): the Hitex logo is printed
User button 2 (SW501): the Texas instruments logo is printed
User button 3 (SW502): the Hitex logo is printed
User button 4 (SW503): the Hitex logo is printed
Reset button (SW300): resets the safety MCU (nRST)

A special function is provided with “user button 3”. Keep this button pressed during Power On of the kit the safety application software stays in a while loop and does no start with the application. This is intended for development purposes to avoid that a miss configuration leads to problems so that a debugger cannot connect anymore.

9.5 LEDs

The kit features several LEDs where some of them are programmable. To get an overview of all the LEDs provided refer to chapter 5.2.3.

The following table lists the application programmable LEDs, along with their drivers

Designator HW	Color	Driver	Designator FW
D501	Blue	Safety MCU	User LED1
D502	Red / Green / Blue	Safety MCU	RGB LED
D503	Blue	Safety MCU	User LED2
D504	Blue	Safety MCU	UserLED3
D505	Blue	Safety MCU	UserLED4
D700	Red	C&M MCU	C&M LED

Table 9-1 User-programmable LEDs

The user LEDs are blinking to indicate activities of the safety application firmware.
User LED1: Indicates communication between safety application and control monitor application.
User LED2: Indicates that the instance retrieving sensor data (temperature and accelerometer) is active.
User LED3: The instance controlling the runtime tests is processing.
User LED4: The external watchdog (Q&A) is in use and serviced.

RGB LED: Green for a successful write of the EPROM data
Red if the TPS can't switch to active state
C&M LED: Is blinking when data communication between C&M device and the GUI takes place.

9.6 User Commands (Template task)

The safety application executes an instance which receives the so called "user commands". The user commands triggered by a user via the GUI are forwarded to the safety application and processed in the template task. This task instance is called template task because it is intended that a user who wants to extend the existing software with own code has a simple entry point to start. The code implemented is posting a text to the onboard display when a command is received.

9.7 Task monitoring

The kit has implemented a task monitoring feature. It is visualized with the GUI in the overview page. The task monitor gives information about the task execution on the safety MCU. More specifically, for each task the start time (in ms), the name and the time that has elapsed since the last execution is given (in us). The latter should correspond to the cycle time during normal operation.

The safety firmware has a call-back function which is executed each time when the scheduler is started. In this call-back routine it is checked if the next executing task is different from the current running one and then a task related information is output via 4 GPIO signals. At least up to 15 tasks could be monitored. In the implementation 8 tasks are monitored. The monitoring application samples the GPIO signals and translates the information to a format suitable for the GUI.

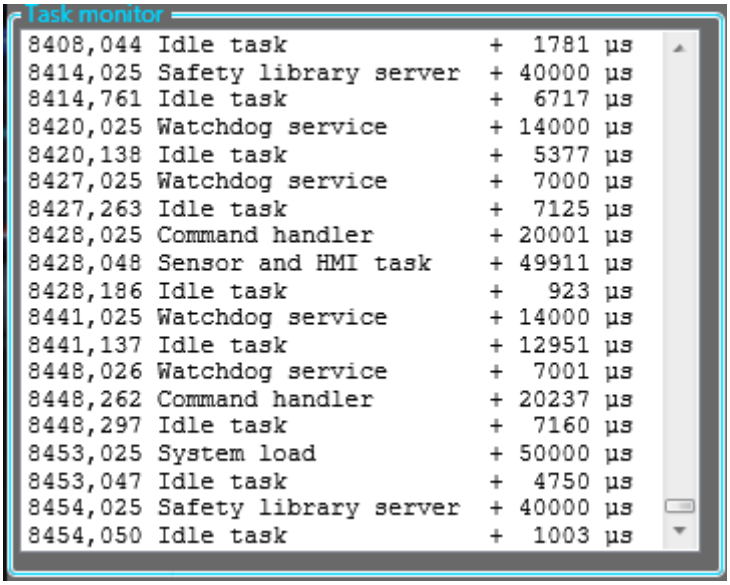
Task monitor
8408,044 Idle task + 1781 µs
8414,025 Safety library server + 40000 µs
8414,761 Idle task + 6717 µs
8420,025 Watchdog service + 14000 µs
8420,138 Idle task + 5377 µs
8427,025 Watchdog service + 7000 µs
8427,263 Idle task + 7125 µs
8428,025 Command handler + 20001 µs
8428,048 Sensor and HMI task + 49911 µs
8428,186 Idle task + 923 µs
8441,025 Watchdog service + 14000 µs
8441,137 Idle task + 12951 µs
8448,026 Watchdog service + 7001 µs
8448,262 Command handler + 20237 µs
8448,297 Idle task + 7160 µs
8453,025 System load + 50000 µs
8453,047 Idle task + 4750 µs
8454,025 Safety library server + 40000 µs
8454,050 Idle task + 1003 µs

Figure 9-3: Task monitor

10 The safety application

This chapter is for users who want to explore the source code of the Safety Application Firmware (SAFW) and modify/extend its functionality. The software frame work is explained along with the major instances and the data flow.

10.1 Considerations before you start

- It is required to accept the license agreement for the SafeTI™-Diagnostic-Library during the installation process. If this is rejected the “Safety application project” does not contain the SDL and therefore cannot be built.

- The monitor application implementation asserts the *ignition* pin of the TPS device. This is necessary for the TPS to start and to deliver the power lines to the safety application. If you remove that in the code then the SDUT gets no more powered. It is implemented in the file “MON_CommandHandler.c” in the function “CMD_Resetappl()”. The function is “MON_ignition()”.

- The safety application checks for a user button pressed at a very early stage of the boot code. If user button SW503 is pressed during power on then the safety app firmware stays in an endless loop. This ensures that reprogramming of the kit is possible even if the current firmware will fail and reset all the time.

- There are different types of kits differentiated by the assembled SDUT MCUs. As the TMS570 controllers support code in big endianess format where the RM48 MCUs support little endianess format, ensure that the correct build configuration is selected in the Code Composer Studio project.

- A compile time configuration setting configured by the C language macro “DEBUG” is defined in “HAS_Config.h”. During debug, it is recommended to set this to 0 – doing this deactivates some interaction with the external watchdog. E.g. this avoids that the ENDRV pin (input from the TPS) monitoring is deactivated.

  Note: Stepping through the code always leads to the ENDRV pin going low.

- A special task is prepared to facilitate a kit user to start development with the kit. For the first simple enhancements the **Template Task** is the right place to insert code.

10.2 Tooling

The firmware is developed with the Code Composer Studio version 5.3.0.00090 (CCS), the development IDE from Texas Instruments based on eclipse. Another tool from TI required is “NOWECC” which is used to calculate the ECC and append it to the output file. These both tools are delivered with the “SafeTI™-HSK Tool DVD. If it is not already done install these tools on your development system (PC). Refer to the tools installation chapter 3.3.
CCS includes a compiler for the Hercules™ Safety MCUs. The compiler version used for firmware development is “TIv5.01”.

Ensure that the Linked Resource “NOWECC” is assigned to the correct installation path.

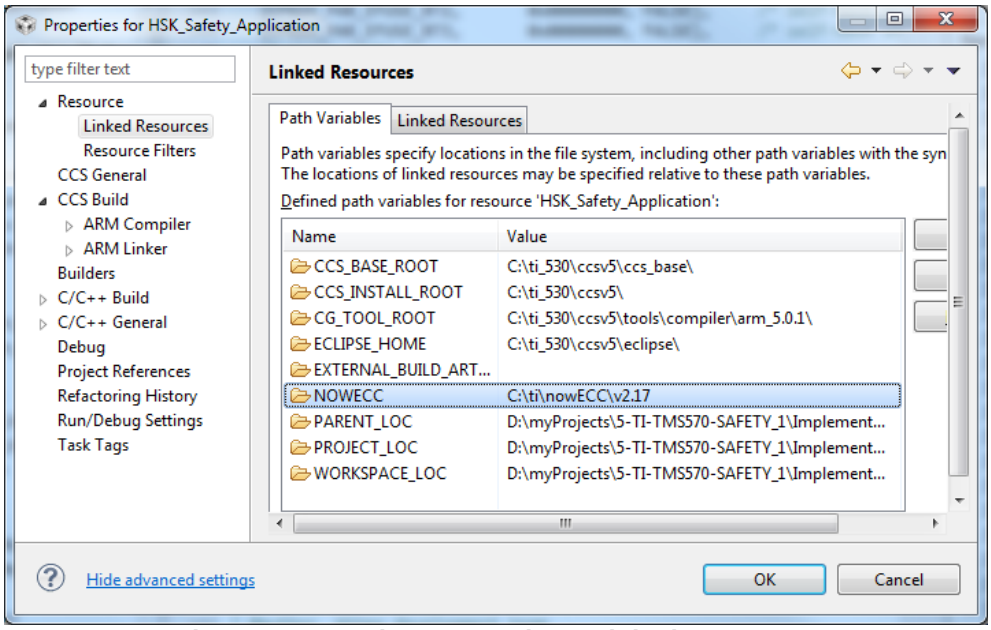

Figure 10-1: Project properties – Linked resources

There are two other tools used in the safety application project but they are optional. One of them is “Subversion” (SVN) for the configuration management and the other one is doxygen for documentation. If you do not use SVN then remove the call from the post build step.

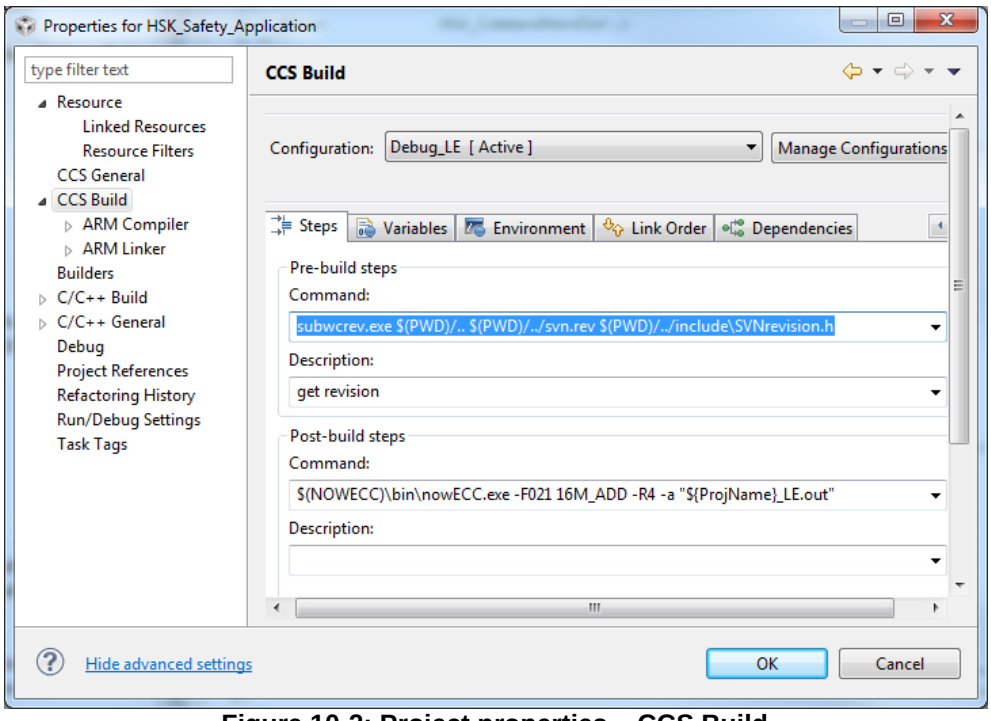

Figure 10-2: Project properties – CCS Build

If not using “doxygen” nothing has to be done at all.

10.2.1 *Import projects into CCS*

Open the “Import” dialogue and select “Existing CCS Eclipse Projects”.

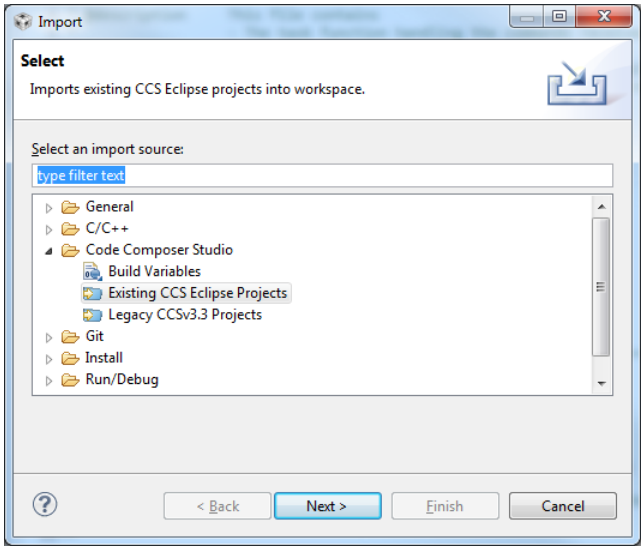

Figure 10-3: Import dialogue

Browse to the kit installation directory and select the safetyApplication.zip.

Directory: …\safeTI-HSK\Firmware-applications\Safety_application\ safetyApplication.zip

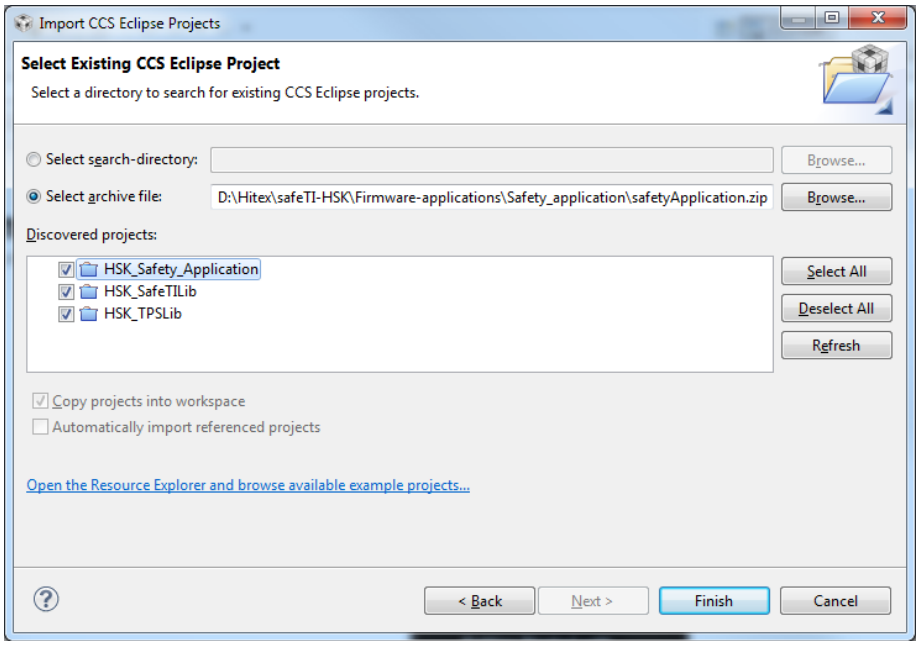

Figure 10-4: Import projects

10.3 Safety application firmware

The SAFW is composed of 3 major parts: the “Safety application”, the “SafeTI™-Diagnostic-Library” and the “TPS-Library”. The source code of the Safety application is completely open source and delivered with an Eclipse projects.
Note: If the license agreement for the SafeTI™-Diagnostic-Library has not been accepted during the installation process, the “Safety application “ project does not contain the SDL and therefore cannot be built.
This description explains only the “HSK_Safety_Application” project.

10.3.1 *Directory structure*

The source code is partitioned to several directories.

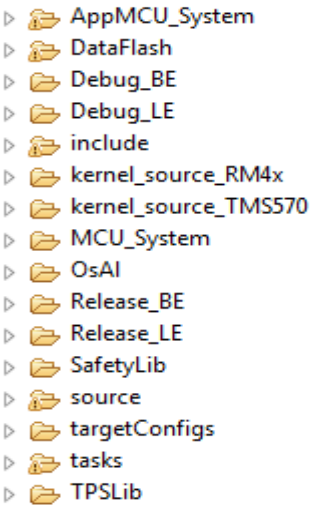

Figure 10-5: Directories

- AppMCU_System
  contains MCU driver files generated with HalCoGen but have been changed manually.
- Include
  this directory contains configuration header files and one file for the subversion revision number.
- Kernel_source_RM48x and kernel_souce_TMS570
  holds the SafeRTOS libraries with the API header.
- MCU_System
  contains MCU driver files generated with Halcogen.
- Osal
  source code for the operating system abstraction layer.
- SafetyLib
  holds the library of the “SafeTI™-Diagnostics Library”.
- Source
  contains the application main.c file and the linker command file.
- Tasks
  contains the sources used by the task instances.

10.4 Architecture

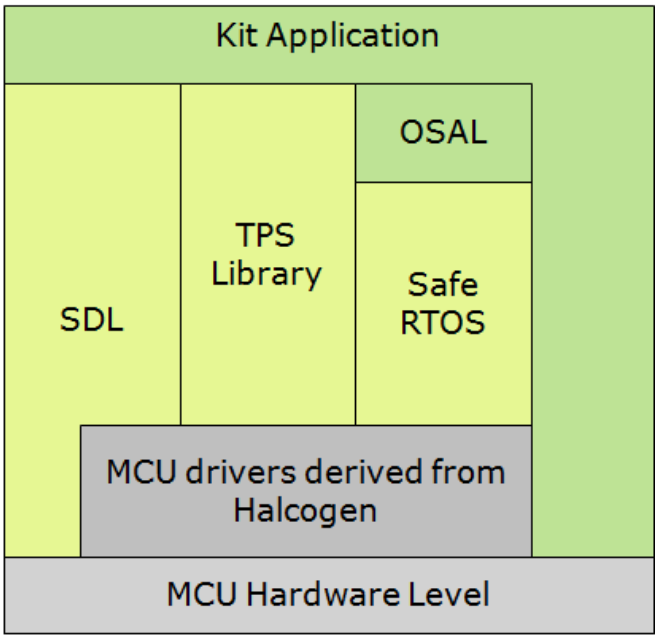

Figure 10-6: SAFW Architecture top level

Components

- MCU drivers:
  Used accessing the peripherals of the MCU (e.g. MIBSPI, ADC, SPI, GPIO, RTI)
- SDL:
  The safety diagnostics library provides a lot of safety diagnostic tests supported by the Hercules hardware controllers. For more information refer to the SafeTI™-Diagnostic-Library documentation.
- TPS Library:
  There is also a library included which provides an API for the handling of the control of the external TPS watchdog device.
- SafeRTOS:
  The kit application is based on SafeRTOS an embedded realtime operating system from Wittenstein.
- OSAL:
  This is an “Operating system Abstraction Layer” .
- Kit Application:
  This is the top level application using all the libraries and layers to fulfill its jobs. (e.g. Injecting faults, Profile measurement). This software will be explained in detail in the following sections.

10.5 Kit application

The kit application’s main responsibilities are partitioned into several task instances which use the inter-task communication mechanisms provided with SafeRTOS.

10.5.1 *SafeRTOS*

The configuration is defined in the SafeRTOSConfig.h

```
#define configMAX_PRIORITIES                    ( ( unsigned portBASE_TYPE ) 10 )
#define configMINIMAL_STACK_SIZE_WITH_FPU       ( ( unsigned portLONG ) 512 )/* Needs to be a power of two value. */
#define configMINIMAL_STACK_SIZE_NO_FPU         ( ( unsigned portLONG ) 256 )/* Needs to be a power of two value. */
#define configTICK_RATE_HZ                      ( ( portTickType ) 1000 )
```

Up to 9 priority levels are supported in this configuration where a value of 9 is the highest possible level. SafeRTOS uses a preemptive scheduling which means that every task could be interrupted anytime by a higher priority task. The tick count is configured to 1 millisecond which means that every millisecond the scheduler is invoked. If no task is ready to run, an idle task will be active.
It has to be considered that some of the tasks run in a privileged mode while others run in an unprivileged mode. An MPU is configured, thus putting restrictions on memory accesses for tasks which run in an unprivileged mode. It is possible to configure up to 4 ranges with specific access rights when creating the task instance.

The example code below illustrates how to configure the access rights:

```
/* configure the mpu according to requirements */
mpuSettings.myPrivilegeLevel = OSAL_PRIVILEGED_TASK;

mpuSettings.myRegionCount = 2;
/* MPU region 0 controls the static data collected in the
 * myCmdUserdata struct */
mpuSettings.myRegions[0].myAccessPermissions = OSAL_MPUACCESS_FULL;
mpuSettings.myRegions[0].myBaseAddress = (void*)&myCmdUserData;
mpuSettings.myRegions[0].myLengthInBytes = CMD_MPU_USERACCESS_AREA_LENGTH;

/* MPU region 1 enables the Peripherals and system memory to access */
mpuSettings.myRegions[1].myAccessPermissions = OSAL_MPUACCESS_FULL;
mpuSettings.myRegions[1].myBaseAddress = (void*)0xF0000000;
mpuSettings.myRegions[1].myLengthInBytes = 0x10000000;
```

For further information on the SafeRTOS refer to the documentation provided by Wittenstein.

10.5.2 *Source files*

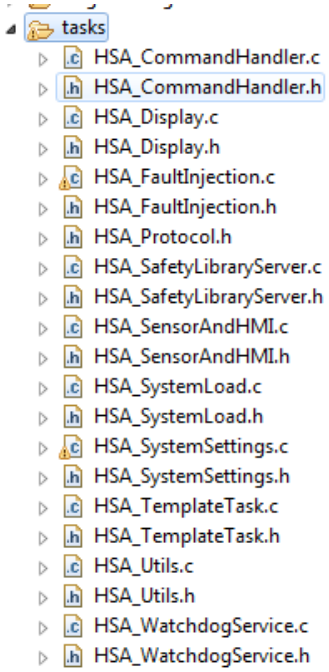

Figure 10-7: kit application source files

10.5.3 *Task overview*

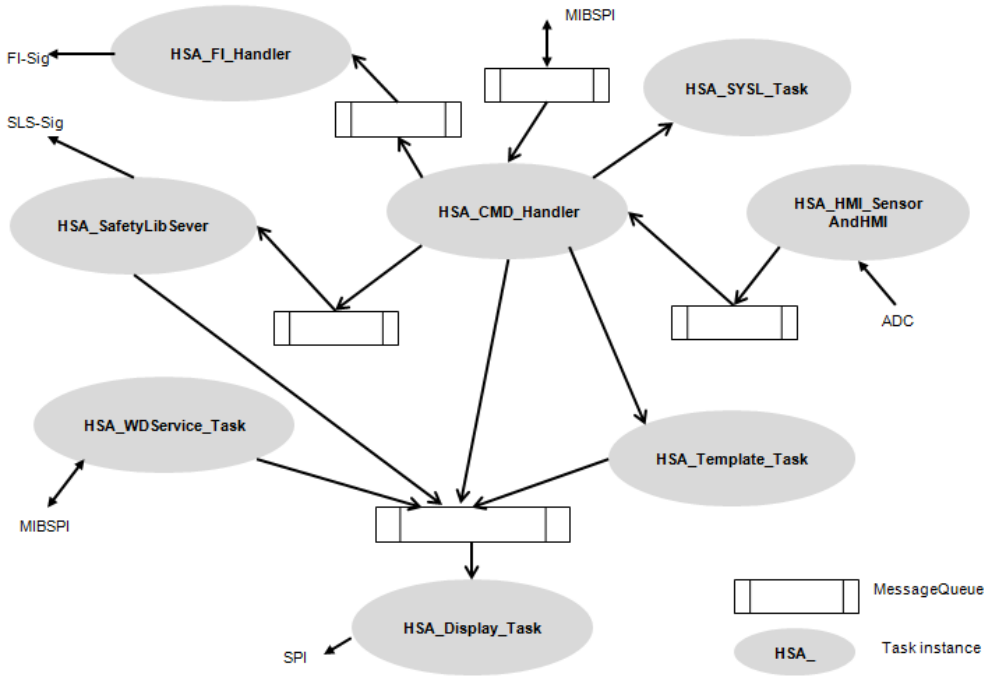

Figure 10-8: Architecture Task Level

The figure above gives a rough overview over the tasks and their communication. The grey ellipses represent one task instance while the white boxes stand for a message queue. Most of the tasks have the structure depicted in Figure 10-9: Generic structure

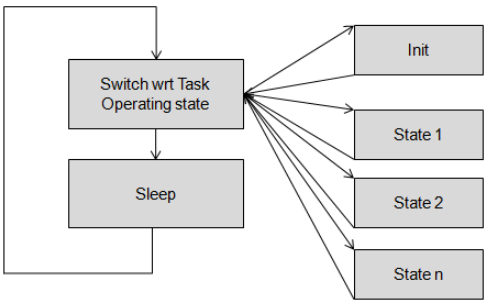

Figure 10-9: Generic structure of tasks in the Safety Kit Application

After the boot phase (see chapter 4.2.1.1) the scheduler starts running the task with the highest priority which is the “HSA_WDService_Task”.

10.5.4 *HSA_CMD_Handler Task*

The responsibility of the command handler task is to process the messages sent from the C&M device. Another job for this instance is to send the application messages (e.g. produced by other tasks) which are available in the *transfer queue* and send these messages to the control and monitor device. These messages could be text messages (“application messages”) or may be sensor data messages containing the acceleration data and the temperature value. When the job is done the task sleeps for a while.
The communication to the C&M device is implemented using the MIBSPI driver. In the receive interrupt handler the received data is put into the “receive external messages” queue using the “OSAL_Queue ReceiveFromISR” routine. To transmit data the DMA methodology (driver provide by HalCoGen) is applied.

Property	
Priority	5
Assigned message queues	5
Assigned LED	LED 1
Privilege level	Privileged
External Interfaces	MIBSPI 3

Table 10-1 Command handler task properties

Message queue	Elements	Type	Counter part
Receive external messages	10	Consumer	C&M device (receive IRQ)
Fault injection	1	Producer	Fault injection task
System load	1	Producer	System load task
User commands	1	Producer	Template task
C&M responses	10	Consumer	All other tasks

Table 10-2 Command handler message queues

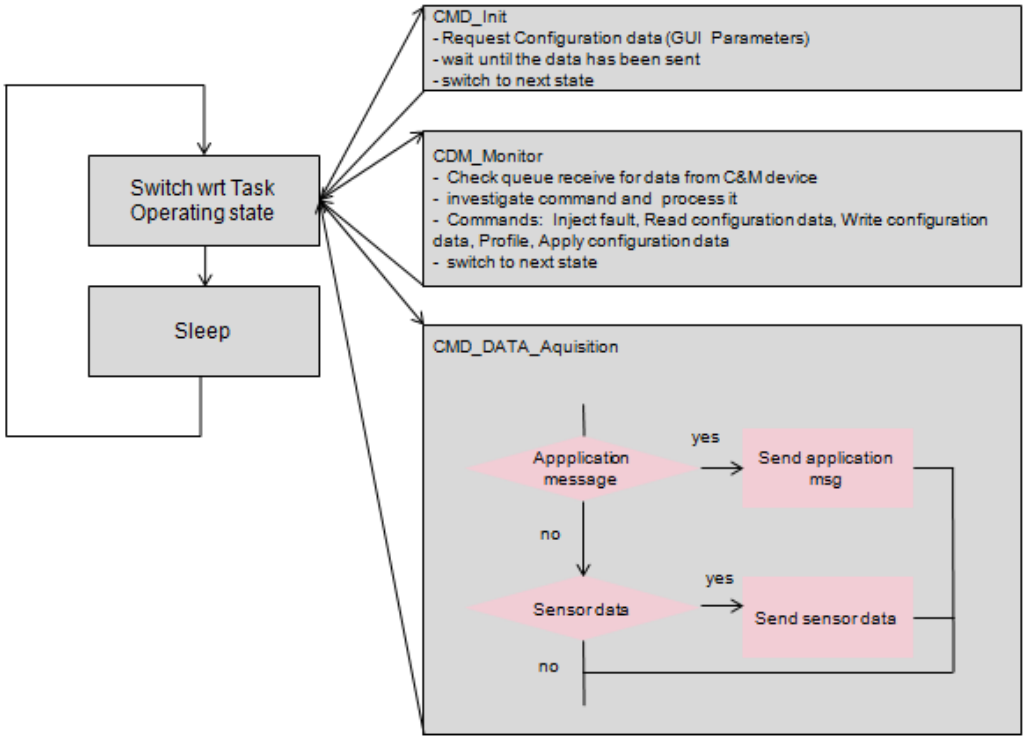

Figure 10-10: Command handler task data flow

10.5.5 *HSA_FI_Handler*

This task instance features the fault injection process. It waits on messages to request a fault injection. If received one the GPIO pin related to the FI signal is asserted. After that the fault is injected which mostly means to call the corresponding function in the SDL. If the function returns the FI signal will be deserted.
It has a very simple structure.

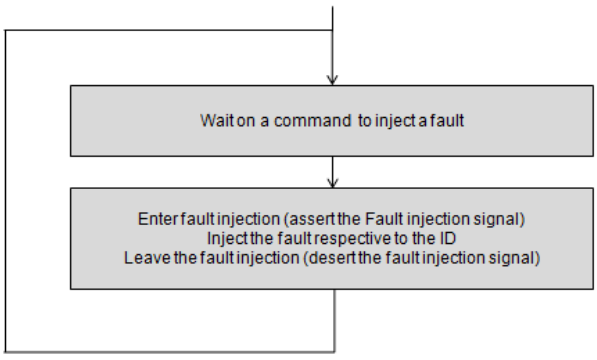
Wait on a command to inject a fault
Enter fault injection (assert the Fault injection signal)
Inject the fault respective to the ID
Leave the fault injection (desert the fault injection signal)

Figure 10-11: Fault Injection handler task data flow

The fault injection routine is a big switch-case construct for the different faults to inject. If a fault is injected very often the system generates an interrupt and enters the “Safe state function”. A parameter setting defines if the system returns back to the calling function or enters a safe state. For most of the faults, the Safety application makes an API call provided by the SafeTI™ Diagnostic Library.

Property	
Priority	8
Assigned message queues	1
Assigned LED	-
External Interfaces	-
Privilege level	privileged

Table 10-3 Fault handler task properties

Message queue	elements	Type	Counter part
Fault injection	1	Consumer	Command handler task

Table 10-4 Fault handler message queues

10.5.6 *HSA_SafetyLibServer*

This task has two main jobs to do:

- One part is to call the cyclic self-tests periodically.
- The other part is to provide profile measurement capability which can be invoked by the user using the HSK Monitor GUI.

The task delay duration depends on the configuration parameter “Safety loop”. This parameter defines the time available for one complete cycle. The variable containing this parameter value is “myCyclePeriod”.

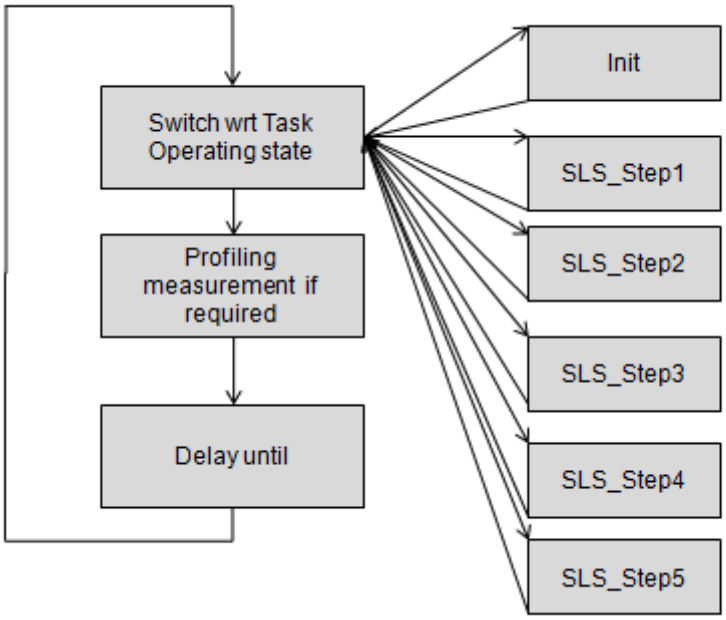

Figure 10-12: Safety library server task data flow

Property	
Priority	4
Assigned message queues	0
Assigned LED	3
Privilege level	privileged

Table 10-5 Safety library server task properties

10.5.6.1 Run time self-test execution

The run time tests executions are implemented in one routine called “HSA_SLS_runCyclicTest”.
As can be seen in the data flow structure in Figure 10-12: Safety library server task data flow, the task has 5 operating states “Step1 to 5”. The safety application can be configured to either run all steps in one go, or to run these individually. A macro defined in the “HSA_Config.h” header configures this functionality:

```
#define RT_ENABLE_SPLIT  FALSE
```

A value of “FALSE” means that the runtime tests are processed in one go. A value of TRUE means to split the execution.
The default value is set to “FALSE”: this executes the safety loop in one go.

Each execution step is assigned a set of self tests. If the self-tests can be selectively enabled or disabled through parameters set in the GUI. The parameter setting is stored in the “HSA_SystemSettings.c” file and applied in a routine in this task instance.

10.5.6.2 Profiling measurement

Before the task finishes its active mode it is always checked if a profile measurement is requested. If so the respective calibration routine is called. In the calibration routine the profiling measurement is prepared and then the “Profile signal” is asserted. Next, the self-test API in the SDL is called. After return from SDL the “Profile signal” is deasserted.

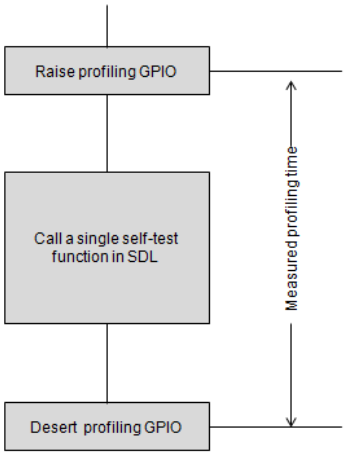

Figure 10-13: Profiling measurement data flow

A special handling is necessary for the “Profiling full safety task” feature. For this case the “Profiling signal” is asserted and then the various selftests configured for all 5 steps are called. After that the signal is deasserted.

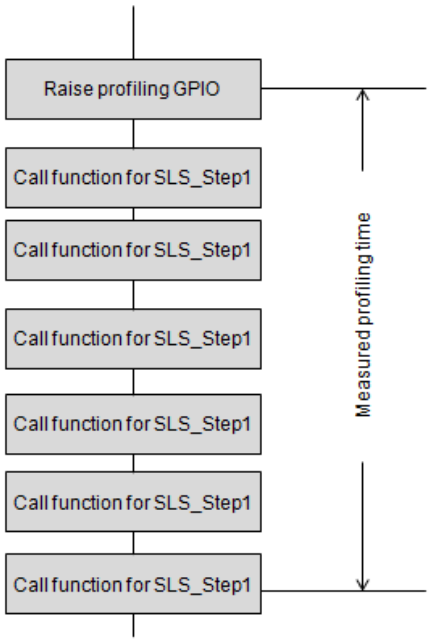

Figure 10-14: Profiling full safety task data flow

10.5.7 *HSA_HMI_SensorAndHMI*

This instance is responsible to collect the accelerometer and the temperature data.

Property	
Priority	2
Assigned message queues	2
Assigned LED	2
External Interfaces	MIBSPI 5 to accelerometer, ADC1
Privilege level	privileged

Table 10-6 Sensor handler task properties

Message queue	elements	Type	Counter part
Sensor data	5	Producer	Command handler task
Display messages	5	Producer	Display task

Table 10-7 Sensor handler message queues

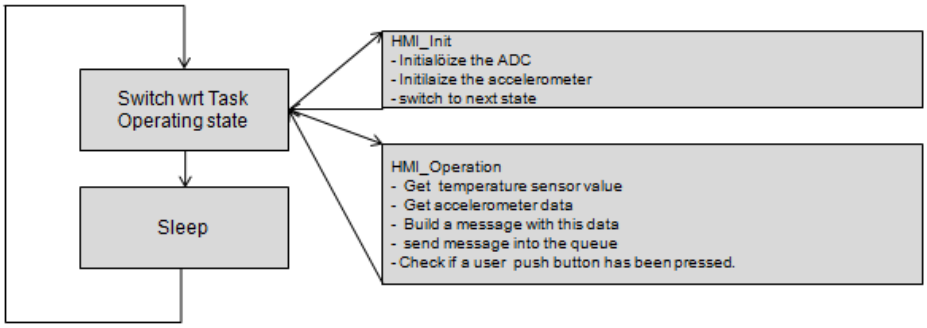
Switch wrt Task
Operating state
Sleep
HMI_Init
- Initialöize the ADC
- Initilsaize the accelerometer
- switch to next state
HMI_Operation
- Get temperature sensor value
- Get accelerometer data
- Build a message with this data
- send messsage into the queue
- Check if a user push button has been pressed.

Figure 10-15: Sensor task data flow

10.5.8 *HSA_WDService_Task*

This task instance is responsible for servicing the external watchdog. When started through the scheduler it waits on the TPS to receive the diagnostic state. After that the TPS startup tests are executed according to configuration. If the tests are successful then the TPS is initialized with respect to the configuration at least deposited in the GUI. When everything is ok the task enters a loop in which the question and answer protocol is operated. It is important for this task to meet the timing constraints specified through the TPS window open/close time. The task uses the SAFERTOS function TaskDelayUntil() to ensure this.

Property	
Priority	Starts with 9 and is changed to 7 in normal operation mode
Assigned message queues	0
Assigned LED	4
External Interfaces	MIBSPI 3 via the TPS library
Privilege level	privileged

Table 10-8 Watchdog server task properties

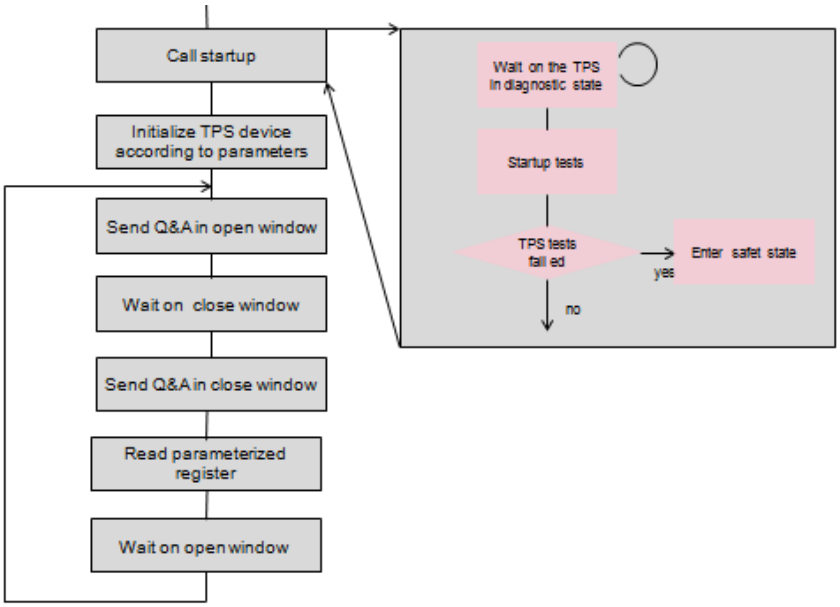

Figure 10-16: Watchdog service task data flow

It is important for this task to meet the timing constraints specified through the TPS window open/close time. The task uses the SafeRTOS function TaskDelayUntil() to ensure this.
For the Q&A protocol the TPS library API is used. After an answer/response cycle the watchdog fails counter is read out and the TPS state is checked. This is required by the TPS sniffer in the C&M device.

10.5.9 *HSA_DisplayTask*

The display instance serves the pictures or texts to be printed to the on-board display. It is realized as a task to ensure that the outputs of texts is serialized and therefore not mixed when different other tasks want to post messages in parallel. The text messages are received out of a queue.

Property	
Priority	2
Assigned message queues	1
Assigned LED	0
External Interfaces	SPI 2 interface to the onboard display
Privilege level	unprivileged

Table 10-9 Display task properties

Message queue	elements	Type	Counter part
Display messages	5	Consumer	All other tasks

Table 10-10 Display message queues

For the data to be output to the display an API is used located in the source file “onboardLCD.c”. The “onboard LCD API” uses a special SPI driver because a special handling with the CS is necessary. This driver is implemented in the source “spi_display.c”.

Source	Type	Comment
Onboard_lcd.c	C source	API implementation
Onboards_lcd.h	Header	API definition
Spi_display.c	C source	SPI driver to display
Spi_display.h	Header	SPI driver interface
BW_LCD.c	C source	LCD command interface implementation
BW_LCD.h	Header	LCD command interface header
Font.h	Header	Character fonts table
Picture.h	Header	Contains the pictures (Hitex and TI logo)

Table 10-11 used files for onboard display api

10.5.10 *HSA_Template_Task*

This task instance is intended for a developer to integrate own code and to extend functionality of the kit. The advantage to add own code here is that the GUI can be used to trigger and control the code execution. No changes in the GUI or in the C&M firmware are necessary.
The data flow of the template task is very simple. It always waits on a user command. If one is received a message is posted to the onboard display. After that it waits again. A board test is implemented as an example how to add own functions.

Property	
Priority	4
Assigned message queues	2
Assigned LED	0
External Interfaces	-
Privilege level	unprivileged

Table 10-12 Template task properties

Message queue	elements	Type	Counter part
User commands	1	Consumer	Command Handler
Display messages	5	Producer	Display task

Table 10-13 Template task message queues

10.5.11 *Parameters*

Few parameters control the behavior of the kit firmware in different situations. So the parameters are not directly assigned to one specific task instance.
The parameters also often named “Configuration Settings” are stored in the HSA_SystemSettings.c source file in an array consisting of structures. The related API functions to manipulate the settings are also implemented in this source file.

```
/*
 * Declaration of a configuration entry
 */
typedef struct
{
                        uint16_t   myAdressId;                          /* address identifier */
                        uint32_t   myValueId;
                        boolean_tmyValid;
}slsConfigEntry;

/*
 * Structure of the complete configuration data set
 */
typedef struct SLSConifgData
{
        /*configuration data for the safety library */
        boolean_t                   mySLSConfigurationHasChanged;
        slsConfigEntry              mySLSCyclePeriod;
        slsConfigEntry              mySLSSystemLoad;
        slsConfigEntry              mySLS_safetyConfigDataESM[COUNT_ESM_SETTINGS];
        slsConfigEntry              mySLS_safetyConfigDataTPS[COUNT_TPS_SETTINGS];
        slsConfigEntry              mySLS_Parameter[COUNT_PAR_SETTINGS];
}slsConfigData;
```

Figure 10-17: Source code extract configuration settings data storage

As you can see each setting has an assigned address as a unique identifier. This address defined in the settings.xml file which is read out by the GUI. It is necessary that the address identifiers used in the SAFW correlate to the ones used by the GUI.

10.5.11.1 Global settings handling

After reset the SAFW starts with the default values of the parameters. The command handler task then requests to the C&M device to send a complete list of the configuration settings. This is necessary to make an alignment with the GUI parameter values. If the parameters are received by the C&M device they are only stored in the array. The parameters take effect after they are applied by an extra command also received from the C&M device.
For the TPS settings the system behaves different. The reason for that is that the “TPS settings” cannot be changed once the TPS has reached the active state.
So the TPS settings are also stored in a data EEPROM. This EEPROM data is read out at the very beginning of the boot phase and the values are copied to the “System settings” storage. Therefore a reset is necessary to make changed “TPS-settings” effective.
For writing and reading the EEPROM data a library from TI is used which supports flashing F021 flashes. When new data is written successfully it is indicated through the RGB LED shining green.

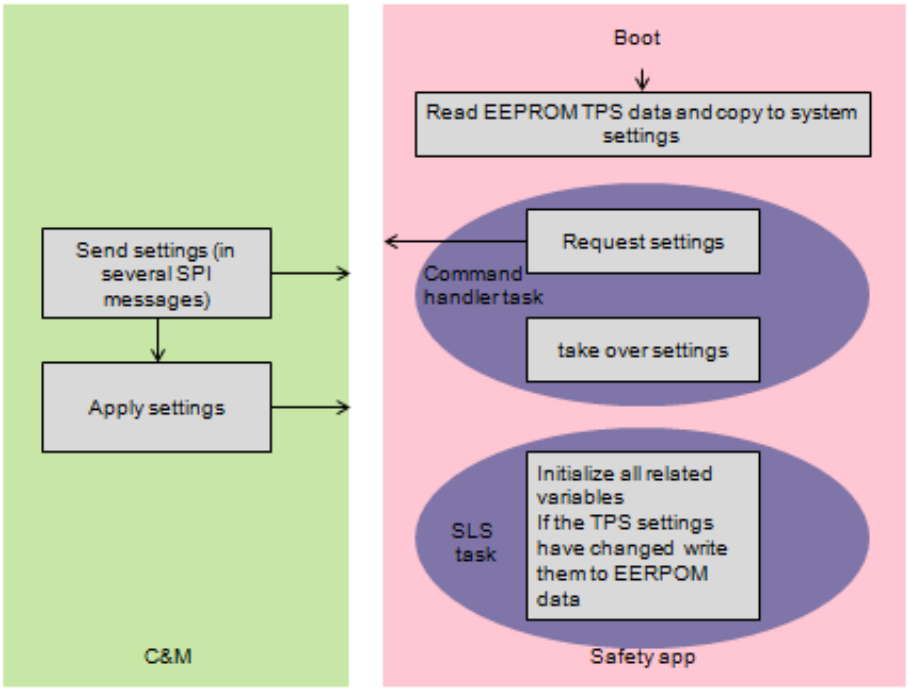

Figure 10-18: data flow of the handling for the Parameter settings

10.5.12 *Typical data flow example for a fault injection*

The rough data flow of a typical fault injection. The example describes a fault injection issued by the SAFW. It is demonstrated for faults which generate an interrupt (ESM or abort).

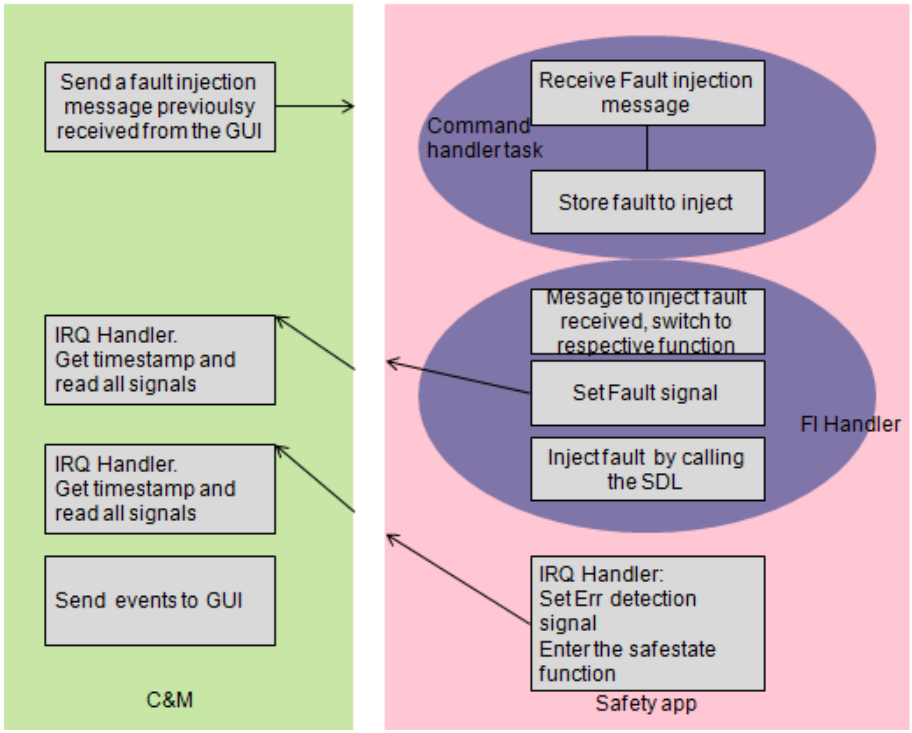
Send a fault injection message previoulsy received from the GUI
Receive Fault injection message
Command handler task
Store fault to inject
Mesage to inject fault received, switch to respective function
IRQ Handler. Get timestamp and read all signals
Set Fault signal
FI Handler
Inject fault by calling the SDL
IRQ Handler. Get timestamp and read all signals
IRQ Handler: Set Err detection signal Enter the safestate function
Send events to GUI
C&M
Safety app

Figure 10-19: data flow of fault injection

10.5.13 *Typical data flow example for a profiling measurement*

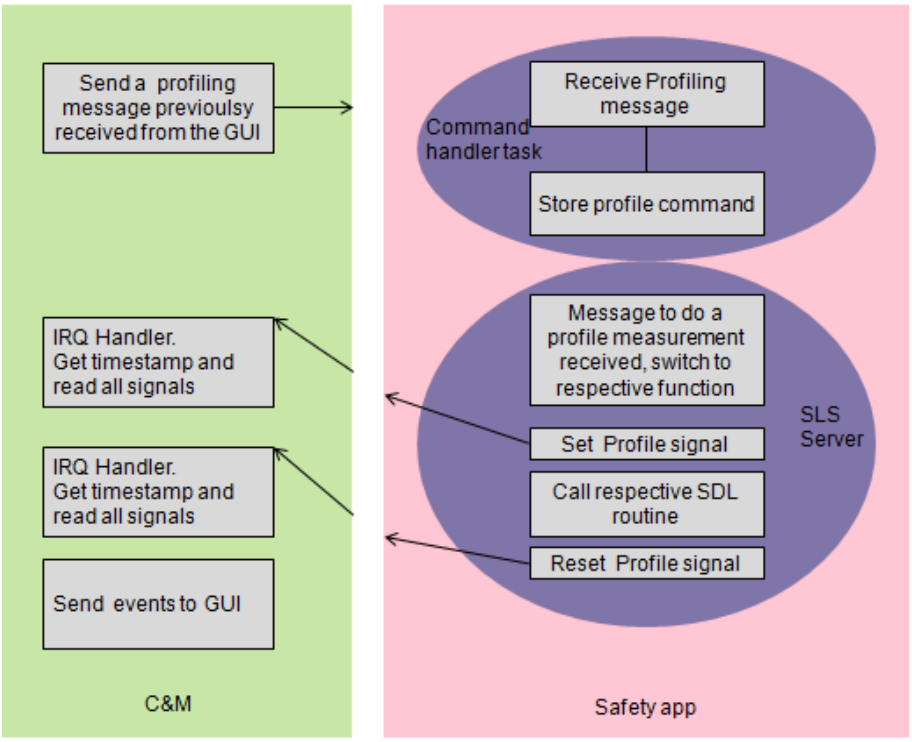
Send a profiling message previoulsy received from the GUI
IRQ Handler. Get timestamp and read all signals
IRQ Handler. Get timestamp and read all signals
Send events to GUI
C&M
Receive Profiling message
Command handler task
Store profile command
Message to do a profile measurement received, switch to respective function
SLS Server
Set Profile signal
Call respective SDL routine
Reset Profile signal
Safety app

Figure 10-20: data flow profiling

11 Troubleshooting

The table below lists some common problems encountered when using the kit.

Symptom	Cause/Workaround
The GUI can't connect to the HSK board even if it is enumerated and indicated on COM port.	Another GUI is already open and connected to the GUI. Only one GUI instance can connect to a HSK-safeTI kit.
The GUI does not show to which COM port the HSK is connected.	Check the FTDI driver configuration on your host.
The red LED (RGB LED) is on.	That means that the TPS didn't operate in active state or does not come up. Maybe a TPS configuration setting is used which does not work properly. Change the settings again and after that power on the kit.
The TPS does not behave as expected after changing the configuration parameters.	Disable recording and enable the recording again. The effect is that the kit gets a reset and takes over the TPS settings.
The XDS debugger can't connect	Power on the kit while keeping the User button SW503 pressed. The try to connect the debugger again. If this does not help close the CCS and restart it.

Table 11-1: Troubleshooting

12 Appendix A: References

[REF_01] 30301-HSK Hardware Specification_V0.1.1.docx
[REF_02] 01003-HSK System Requirements_V1.1.docx

w w w . h i t e x . c o m

Published by Hitex Development Tools GmbH

40100 User Manual, V1.2